НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки “

на тему “ Розробка системи обліку та аналізу науково-дослідних робіт “

Виконав (-ла): студент (-ка) 4 курсу, групи ТР-51

Бувалець Анатолій Юрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент , к.т.н., Лабжинський В.А. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.т.н. Сенченко В.Р. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту
Бувальцю Анатолію Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Розробка системи обліку та аналізу науково-дослідних робіт“

керівник роботи доцент Лабжинський В.А.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”____ 201__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Мова програмування C# у середовищі VisualStudio 2017 у вигляді API-сервісу за технологією ASP.NET WebAPI

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) Дослідити існуючі програмні засоби для обліку результатів наукової діяльності. Розробити алгоритм та реалізувати програмний продукт для обліку договорів науково-дослідних робіт.

5. Перелік ілюстративного матеріалу
Архітектура системи, функціональні можливості системи, структура даних, інструкція з використання програмного продукту

6. Дата видачі завдання ”__”_____201__р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	01.12.2018-31.03.2019	
2	Розробка архітектури та загальної структури системи	01-18.04.2019	
3.	Розробка структур окремих підсистем	19-25.04.2019	
4.	Програмна реалізація системи	26.04-13.05.2019	
5.	Оформлення пояснювальної записки	06.05-01.06.2019	
6.	Захист програмного продукту	25.05.2019	
7.	Передзахист	01.06.2019	
8.	Захист	18.06.2019	

Студент _____ Бувалець А.Ю.

(підпис) (прізвище та ініціали,)

Керівник роботи _____ Лабжинський В.А.

(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Обсяг дипломної роботи складає 69 сторінок, 23 рисунки, 8 таблиць та 25 посилань.

Метою дипломної роботи є розробка програмної системи для оптимізації роботи над збором даних про процес виконання договорів НДР та полегшення процесу аналізу введених даних.

Було проведено аналіз систем, які мають схожі з створюваною системою функціональні можливості, описано їх переваги та недоліки.

Результатом роботи є веб-додаток, створений для забезпечення можливості проведення обліку та аналізу договорів робіт. Детально описано покрокову роботу з системою. Також описано процес інсталяції системи та зазначено розподіл ролей в системі.

Ключові слова: науково-дослідна робота, облік науково-дослідних робіт, документообіг.

ABSTRACT

The total amount of work is 69 pages, 23 figures, 8 tables and bibliography of 25 references.

The purpose of the graduate work is to optimize the work of collecting data on the process of implementing the agreements on research and development and to facilitate the process of analysis of the data entered.

An analysis of systems that are similar to the functionality created by the system was analyzed, their advantages and disadvantages were compared.

The result of the work is a web application, created to provide the opportunity for accounting and analysis of research contracts. Detailed step-by-step work with the system is described. Also described is the process of installing the system and distributing roles in the system.

Key words: research work, accounting of research works, document accounting.

ЗМІСТ

ВСТУП 8
1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ 11
1.1 Інтерфейс створення даних 11
1.2 Забезпечення гнучкого перегляду договорів 11
1.3 Інтерфейс для полегшення аналізу процесу виконання договорів 12
1.4 Забезпечення безпеки даних 13
2 АНАЛІЗ ПРОБЛЕМИ СТВОРЕННЯ СИСТЕМ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ 14
2.1 Опис існуючих систем 15
2.2 Аналіз існуючих систем, схожих за функціональними можливостями до розроблюваної 19
3 ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ 20
3.1 Засоби для створення системи 20
3.2 Опис засобів для реалізації серверної частини 21
3.3 Опис засобів реалізації клієнта 22
3.4 Опис середовищ розробки 24
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ 26
4.1 Архітектура системи 26
4.2 Опис функціональних можливостей системи 30
4.3 Діаграма класів 32
4.4 Модель бази даних 34
4.5 Опис таблиць бази даних 35
5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ 39
5.1 Інсталяція та системні вимоги 39
5.2 Інструкція з використання програмного продукту 39
ВИСНОВКИ 49

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТОК А 53
ДОДАТОК Б 55
ДОДАТОК В 63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

НДР — науково-дослідна робота

ASP.NET (Active Server Pages для .NET) — технологія створення веб-застосунків і веб-сервісів від компанії Майкрософт

API (Application Programming Interface) — інтерфейс прикладного програмування

REST (Representational state transfer) — підхід до архітектури мережевих протоколів

JSON (JavaScript Object Notation) — формат даних

HTTP (HyperText Transfer Protocol) — мережний протокол передачі даних

ORM (Object-relational mapping) — технологія для пов’язування бази даних з концепціями об’єктно-оріентованих мов програмування

DI (Dependency Injection) — шаблон для передачі зовнішніх залежностей окремому програмному компоненту (одна з форм інверсії управління)

БД — база даних

СКБД — Система керування базами даних

MS SQL (Microsoft SQL Server) — СКБД, що розповсюджується корпорацією Microsoft

ВСТУП

Однією з найважливіших функцій науково-дослідницьких інститутів та кафедр є науково-дослідницька діяльність. Однак, ведення обліку виконання укладених договорів по НДР є застарілим, методи збору даних є неефективними та результати їх не мають інтуїтивного відображення результатів. Щоб створити звіт для перевірки виконання договорів НДР потрібно вручну збирати інформацію про кожен договір та вручну формувати звіт. Такий підхід не є ефективним, займає багато часу та не надає бажаного результату.

Результати наукової та навчально-методичної діяльності університету є найважливішою складовою його успішності і конкурентоспроможності на ринку освіти. Над проблемою підвищення результативності та ефективності діяльностей, в останні роки працюють не тільки різні державні інстанції, але співробітники різних вузів.

Рівень цілісного освітнього процесу у ВНЗ в більшій мірі залежить від його організації, профорієнтаційної роботи, розроблених навчальних планів, контролю і оцінки якості освіти і т.п. Особлива увага приділяється кваліфікаційним вимогам до співробітників, в яких виділені показники навчально-методичної та наукової роботи. В основному ці види діяльності відносяться до викладачів вузу і спрямовані на підвищення рівня їх науково-дослідницької діяльності, виховання і кваліфікації майбутніх фахівців.

В даний стали популярними впровадження та створення власних системи для обліку наукової діяльності співробітників, якому у кожного з них є персональна сторінка (особистий кабінет), на якій він може опублікувати професійні дані, навчальні курси тощо, а також додати або редагувати наукові публікації. В деякі звіти інформація про публікації береться з бази даних цього порталу, але інформація про навчально-методичної діяльності співробітника не використовується для побудови звітів про дану роботу співробітників кафедри і підрозділи. За цим стоїть проблема

поганої організації збору та підготовки даних для формування звітів, тому що до упорядника звітів доходять неструктуровані дані. Це тягне за собою ще одну проблему, таку як висока трудомісткість роботи з обліку навчально-методичної діяльності. До того ж, форма звіту щорічно переглядається і можливі зміни її структури, додаються або виключаються пункти з наукової або навчально-методичної діяльності співробітника або змінюються параметри робіт, що говорить про не відсутність статичності структури звіту.

Для оптимізації роботи з договорами необхідно створити інформаційну систему для обліку робіт та швидкої перевірки процесу їх виконання. Така система дозволить автоматизувати процес обліку та полегшити процес перегляду процесу виконання договорів. Також система дозволить прикріпляти до виконуваних договорів документи та завантажувати їх миттєво.

Перевагами впровадження подібної електронної системи для обліку договорів є:

— швидке та зручне керування даними;

— облік може проводитись безпосередніми керівниками або виконавцями без долучення сторонніх осіб;

— інтуїтивне представлення необхідної інформації;

— доступ до даних є цілодобовим;

— робота з інформацією можлива для декількох авторизованих користувачів;

— прикріплення та завантаження існуючих документів, пов’язаних з договорами;

— перенесення термінів виконання є інтуїтивним та його зміна здійснюється миттєво в базі даних.

Записка складається з 5 розділів:

В першому розділі описується призначення системи обліку та аналізу НДР та задачі, які розв’язуються системою.

В другому розділі описується аналіз предметної області (існуючих аналогічних програмних систем для задачі обліку укладених договорів з НДР).

В третьому розділі описуються засоби розробки системи та програмні середовища розробки.

В четвертому розділі описується програмна реалізація системи, архітектура системи.

В п’ятому розділі описується робота користувача з системою та вимоги до інсталяції системи.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ

Призначення розроблюваного програмного забезпечення полягає в використанні його в науково-дослідницьких інститутах для оптимізації роботи над збором даних про процес виконання договорів НДР та полегшення процесу аналізу введених даних. Для надання повної та актуальної інформації з виконання договору необхідно гарантувати можливість оперування етапами виконання договорів та їх інтуїтивне відображення в таблиці для легкого аналізу відсотку виконання для кожного з договорів окремо. Задачі, які мають бути виконані розробленою системою будуть наведені нижче.

1.1 Інтерфейс створення даних

Основною задачею, що має забезпечити нормальне функціонування системи обліку є забезпечення користувача інтерфейсом для створення необхідних даних в базі за вже існуючими документами. Обсяг даних, які вводить користувач не має бути надмірним, але повністю задовольняти всі необхідні вимоги при перегляді. Також необхідно забезпечити користувачу інтерфейс для прикріплення за договором всіх необхідних документів та завантаження їх. Інтерфейс має бути простим і в користувача не має виникати проблем при роботі з ним.

1.2 Забезпечення гнучкого перегляду договорів

Для спрощення аналізу виведених даних, необхідно забезпечити для користувача максимально простий, доступний та повний перегляд введених даних з можливістю їх редагування та видалення. Для забезпечення простого аналізу виведених даних необхідно реалізувати інтерфейс для гнучкого перегляду і фільтрації

даних. Для пришвидшення навігації для користувача необхідно гарантувати всі можливі методи для пошуку та фільтрації наданої йому інформації.

1.3 Інтерфейс для полегшення аналізу процесу виконання договорів

Основним призначенням створеного програмного забезпечення є створення інтерфейсу для полегшення аналізу договорів та процесу їх виконання. Дані мають сортуватись за певними критеріями, в результаті чого буде створено звіт з процесу виконання договорів для всього набору та виконання етапів окремих договорів за відповідним критерієм.

Система має надавати можливість перегляду необхідних даних за наступними критеріями:

— часовий проміжок;

— організація-виконавець;

— організація-замовник;

— безпосередні виконавці та керівники.

Результат виконання подається у вигляді таблиці договорів. Кожен пункт договору містить вкладений список його етапів. Якщо критерієм є часовий проміжок, результат подається у вигляді часового проміжку з нанесеними на нього етапами виконання для відповідних договорів за вказаними на них датами початку та планованого завершення робіт. Для етапів виділено статус їх виконання, дати початку та планованого завершення робіт, їх вартість та фактичну оплату. Статус виконання визначається фактом та вчасністю його виконання. Етапи з заборгованостями мають бути виділені. Необхідно вказати відсоток виконання кожного договору та договорів вказаної вибірки в цілому. Заборгованими вважаються договори та їх відповідні етапи, які не є виконаними після моменту настання запланованого граничного терміну.

1.4 Забезпечення безпеки даних

Права вносити зміни до БД системи мають бути лише в авторизованих користувачів. Для захисту даних необхідно організувати відповідну авторизацію. Доступ до даних не має бути вільним, необхідно мати обмежений список зареєстрованих облікових записів, яким надано доступ до системи та поширити дані, які є необхідними для доступу до системи між безпосередніми її користувачами.

Для забезпечення цього необхідно створити функціональні можливості для адміністратора, якому буде доступні створення та оперування обліковими записами всіх користувачів.

В системі має бути доступно 2 ролі: адміністратор та звичайний користувач.

Адміністратору має бути доступний функціонал оперування обліковими записами, користувачу – створення, редагування облікової інформації та її перегляд.

2 АНАЛІЗ ПРОБЛЕМИ СТВОРЕННЯ СИСТЕМ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ

Систем, які б повністю задовольняли всім вимогам та виконували б всі поставлені задачі немає, але існують системи, які мають функціональні можливості, близький до необхідного. Автоматизованим системам обліку та інвентаризації результатів наукової діяльності співробітників науково-дослідницьких інститутів та кафедр зараз приділяється багато уваги. В деяких університетах розроблено і впроваджено власні системи обліку результатів науково-дослідницьких робіт [1]. Також схожі функції можуть виконувати системи для управління проектами. Програмне забезпечення для управління проектами використовується, щоб допомогти віддаленим командам відстежувати важливі елементи робочого процесу, які включають

— опис завдання;

— відповідальні та виконавці завдання;

— порядок важливості завдання;

— обговорення питань, які можуть виникнути в процесі робіт [2].

Для гарантування вчасного виконання проекту необхідно, щоб всі учасники працювали в команді, кожному учаснику відома його робота і таким чином виконання проходить синхронізовано. Це є особливо актуальним для команд, учасники яких працюють віддалено або незалежними групами. Без відповідного інструменту для синхронізації роботи виконання проекту сильно ускладняється. Це особливо стосується великих проектів.

Система управління проектами надає керівнику проекту можливість повністю скласти структуру проекту та призначити на виконання його етапів відповідних членів команди. Після призначення завдань кожен виконавець проекту бачить призначені йому завдання, що пришвидшує та оптимізує роботу виконання проектів командами.

2.1 Опис існуючих систем

Було проведено аналіз на прикладі таких програмних систем:

1. Автоматизована система обліку результатів інтелектуальної діяльності наукової організації

Система передбачає вирішення двох основних завдань: зберігання інформації та організація доступу до неї. Водночас необхідно враховувати завдання в контексті користувачів системи або їхніх потреб. В даний час інформаційна система в існуючій реалізації автоматизує роботу дослідників, співробітників бібліотеки та звітних працівників [3].

Функціональні можливості системи

Дослідники мають можливість:

— реєструвати в бібліотеці дані про власні публікації, редагувати введені дані;

— отримувати звітну інформацію про внесені в систему ним публікації (рисунок 2.1).

Працівники бібліотеки мають можливість:

— здійснювати облік публікацій;

— затверджувати внесені зміни до списків видань, що були здійснені іншими користувачами;

— отримати звітну інформацію про авторів;

— отримати звітну інформацію про публікації відповідного автора.

Начальники наукових підрозділів мають можливість:

— переглядати звітні дані про публікації співробітників власного підрозділу, проводити створення та редагування цих даних;

— отримати звітну інформацію про публікації відповідного наукового підрозділу наукових підрозділів.

Технічні аспекти системи

Однією з особливостей розробленої системи є те, що вона не вимагає встановлення на комп'ютері спеціального програмного забезпечення. Вам потрібен

доступ до Інтернету та браузер, доступний на всіх комп'ютерах. Фактично весь системний програмний код і БД знаходяться на сервері, клієнтська частина - це браузер, який ініціює запити до сервера і відображає результати його роботи.

Переваги підходу клієнт-сервер:

— підтримка будь-якого апаратного забезпечення (включаючи сильно застаріле) доки системою підтримується браузер;

— оновлення системи відбуваються на сервері, незалежно від клієнта. Тому клієнт не приймає ніякої участі в оновленнях та завжди працює з останньою версією системи.

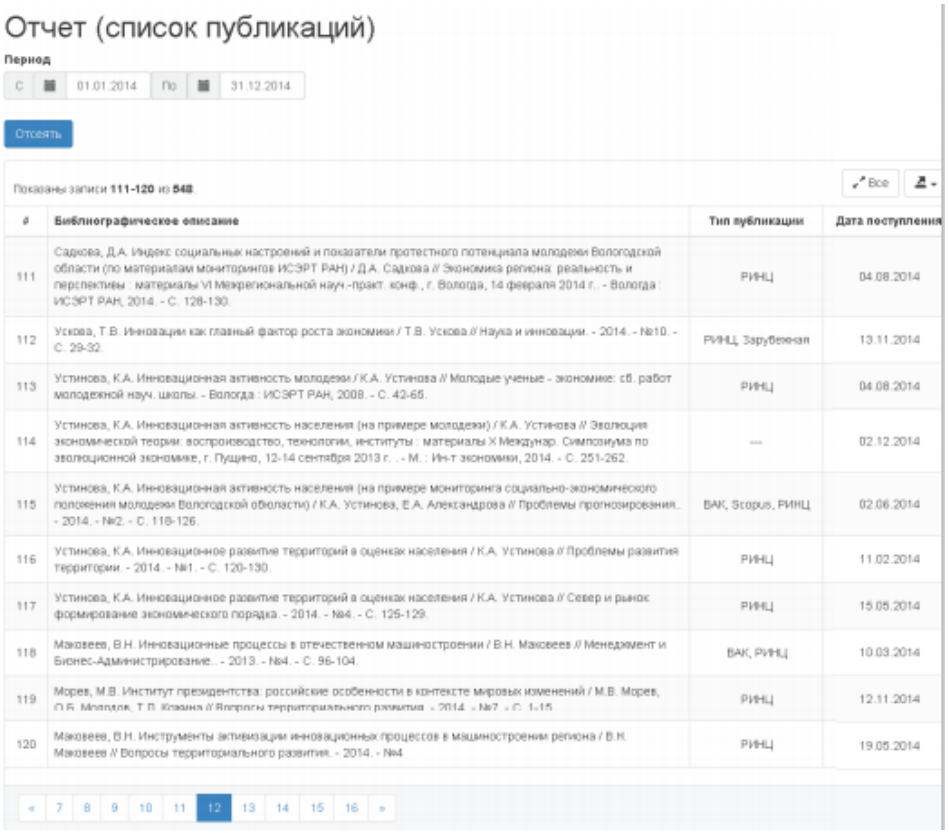

Рисунок 2.1 – Список публікацій

Користувачу може бути присвоєно одну або декілька ролей одночасно (керівник підрозділу та дослідник).

Робота з системою представляє собою роботу над двома її модулями: користувача та публікації.

Користувацький призначений для процесів авторизації та автенфікації. За його допомогою визначається роль користувача в системі, за допомогою чого йому надаються різні функціональні можливості.

У модулі, призначеному для роботи з публікаціями реалізовано основну функціональність системи: облік видань (те деякими ролями – перегляду та редагування інформації про авторів). Звітна інформації про публікації подається у вигляді списку. Для ролі “Дослідник” доступні для перегляду лише власні публікації, доступне їх редагування та відправлення їх в бібліотеку для подальшої верифікації змін. Працівник бібліотеки бачить всі публікації (рисунок 2.2), він може переглядати, додавати, редагувати публікації і підтверджувати їх зміни. Також для цієї ролі реалізовано інтерфейс для перегляду видань за створених відповідними авторами, де відображається весь список авторів, і для кожного з них можна побачити їх список видань.

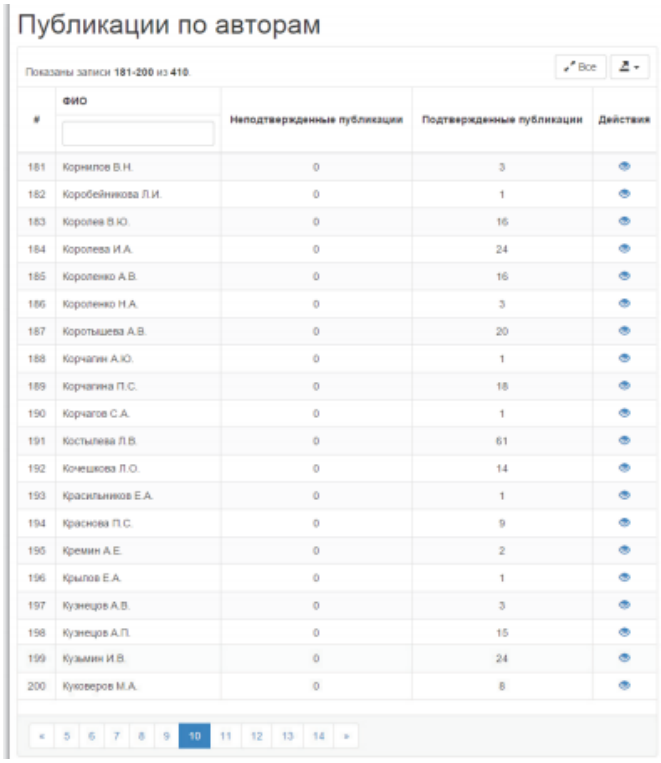

Рисунок 2.2 – Інтерфейс публікацій за авторами

Для кожного з інтерфейсів передбачено фільтрацію даних, наприклад, за датою публікації [3].

2. Системи управління проектами, як приклад – Microsoft Project

Microsoft Project була створена корпорацією Microsoft.

Система призначена для забезпечення підтримки керівників при плануванні робіт над проектом, розподіленні ресурсів, слідкуванні за прогресом виконання, управлінні бюджетами та аналізі робочих навантажень.

Створені плани робіт відображаються в системі на діаграмах Ганта (рисунок 2.3).

Доступні ресурси з загального пулу ресурсів призначаються відповідним проектам. Також доступні інструменти для перегляду часової діаграми розподілених ресурсів для кожного з проектів, що полегшує роботу з пулом ресурсів та їх розподіленням між проектами. Ресурсами можуть бути люди, матеріали чи обладнання. Всі вони мають бути внесені в пул ресурсів. Є можливість призначення для частини проекту кілька ресурсів та навпаки [4].

Бюджети в системі визначаються на основі робіт та ставок ресурсів. Оскільки ресурси призначені для виконання завдань і оціночної роботи призначення, система проводить обчислення вартості, що дорівнює частоті робочого часу. Вона переноситься на рівень завдання, потім до будь-якого зведеного завдання і, нарешті, до рівня проекту.

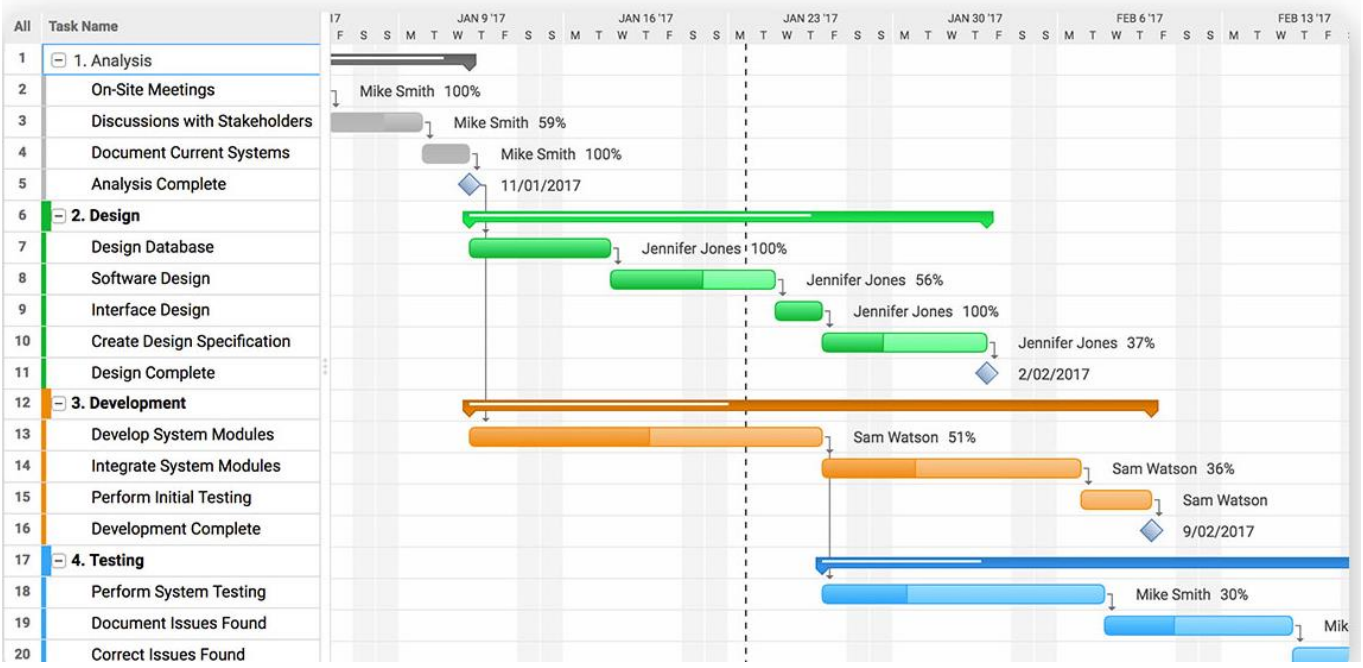

Рисунок 2.3 – Інтерфейс Microsoft Project

Для системи доступно визначення різних ролей користувачів. Це дозволяє розділити та обмежити доступ до функціональних можливостей роботи над проектом для різних ролей користувачів [4].

Календарі, перегляди, таблиці, фільтри та поля, зберігаються в корпоративній глобальній БД, що гарантує можливість роботи з ними для всіх користувачів.

2.2 Аналіз існуючих систем, схожих за функціональними можливостями до розроблюваної

Для систем, впроваджених в університетах, недоліком відносно заданих роботою задач є акцент уваги на персональних та вже виконаних НДР, недостатній функціонал роботи з науковими роботами поетапно для відслідковування виконання договорів, громіздкий, неінтуїтивний інтерфейс користувача.

Системи управління договорами надають можливість для нанесення етапів виконання договорів та їх статусів, вказання відповідальних за них осіб. Але ці системи не призначені для управління договорами НДР. Вони не надають необхідної звітової інформації, яка має надаватись розроблюваною системою. Розроблена програмна система дозволить здійснювати облік робіт в процесі їх виконання, перевіряти статус виконання та переглядати плановані терміни завершення етапів договорів та виникнення заборгованостей.

3 ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ

Дуже важливим кроком при плануванні розробки системи є вибір доцільних програмних засобів, які б могли задовільнити вимоги користувача і полегшити сам процес реалізації системи.

3.1 Засоби для створення системи

При створенні системи було використано такі засоби:

Серверна частина:

— мова програмування C# платформи .NET як мову написання веб-застосунків серверної частини;

— ASP.NET WebApi 2 як фреймворк для написання веб-додатків;

— архітектурний стиль REST;

— фреймворк Autofac як основний DI-контейнер.

Клієнтська частина:

— мова програмування TypeScript;

— фреймворк Angular як основна платформа для розробки клієнтської частини системи;

— HTML як мова розмітки сторінок;

— CSS та Bootstrap для стилізації.

Для серверної частини для системи було обрано операційну систему Windows, для використання клієнтської частини необхідний браузер, незалежно від операційної системи.

Основними середовищами розробки були Microsoft Visual Studio 2017 для серверної частини та JetBrains WebStorm для клієнту. Також для роботи зі створеною БД було використано СУБД MS SQL.

3.2 Опис засобів для реалізації серверної частини

Веб-додаток було створено з використанням ASP.NET. ASP.NET – це веб-платформа для створення веб-сайтів і веб-додатків з використанням HTML, CSS і JavaScript. Є також можливість створювати API і використовувати технології реального часу, такі як веб-сокети.

ASP.NET надає три інструменти для створення веб-додатків: Web Forms, Web Pages та ASP.NET MVC. Інструменти є стабільними, і можна створювати веб-програми з будь-яким з них [5].

Кожна структура спрямована на свій стиль розвитку. Інструмент, що ви маєте обрати, залежить від комбінації ваших програмних засобів, типу програми, яку ви створюєте, та підходу, який вам зручно використовувати [5].

API було створено за допомогою WebAPI. ASP.NET WebAPI – це фреймворк, що призначений для написання API, тобто служб на основі протоколу HTTP на платформі .NET Framework.

В більшості випадків фреймворк використовується для створення служб RESTful.

Основна функція та обмеження, які накладає ASP.NET WebAPI часто використовуються для створення додатків за архітектурним стилем REST, WebAPI не накладає обмежень на створення додатків, які не використовували б вказаний стиль [6].

REST – це стиль розробки архітектури програмного забезпечення, що накладає на розробку системи певні обмеження, широко використовується для створення API. Для обміну інформації використовується HTTP.

Обмеження, які накладає на розробку API стиль REST:

— клієнт відправляє запит, а сервер надсилає відповідь. Такий поділ підтримує незалежність та еволюцію логіки клієнта та логіки на стороні сервера.

— на сервері не повинна зберігатися інформація, пов’язана з клієнтом. Запит клієнта повинен містити всю інформацію, необхідну серверу для обробки цього запиту. Це гарантує, що кожен запит обробляється сервером самостійно.

— використання кешу оптимізує роботу клієнта з сервером, дозволяючи уникнути повторного звертання до серверу [6].

— єдиний інтерфейс визначає інтерфейс між клієнтом і сервером. Використовуються визначені об’єкти класів як ресурси і HTTP-методи – GET, PUT, POST і DELETE. Ресурс ідентифікується URI (Uniform Resource Identifier)

3.3 Опис засобів реалізації клієнта

Клієнтську частину було реалізовано мовою програмування TypeScript, розробленій та підтримуваній корпорацією Майкрософт.

Код експериментального компілятора, котрий транслює код TypeScript в представлення JavaScript, поширюється під ліцензією Apache, розробка ведеться в публічному репозиторії через сервіс CodePlex. Специфікації мови відкриті і опубліковані в рамках угоди Open Web Foundation Specification Agreement (OWFa 1.0).

TypeScript є зворотно сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js. TypeScript має синтаксис, дуже схожий на JavaScript, основною різницею є статична типізація. TypeScript багато в чому базується на специфікаціях наступної версії JavaScript, що розробляється комітетом ECMA .

Підтримка динамічної типізації зберігається — компілятор TypeScript успішно обробить не модифікований код на JavaScript. Основний принцип мови — весь існуючий код на JavaScript сумісний з TypeScript, тобто в програмах на TypeScript можна використовувати стандартні JavaScript-бібліотеки і раніше створені напрацювання. Можна залишити існуючі JavaScript-проекти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли,

які не заважатимуть розробці і прямому використанню проекту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

Функціональні можливості, що надає TS:

— система для роботи з модулями та класами;

— успадковування інтерфейсів та класів (включаючи множинне успадкування);

— опис власних типів даних;

— створення загальних інтерфейсів;

— задання типу змінної (або її інтерфейсу);

— задання сигнатури методів [8].

Фреймворком для розробки клієнтської частини системи було обрано Angular. Angular (підтримуваний Google) є платформою для розробки програмного забезпечення (у відкритому доступі), яка використовується для побудови інтерфейсів користувача.

Плюси Angular:

Архітектура проекту будується компонентами, що підвищує якість коду. Компоненти – це незалежні частини системи. Структура проекту Angular схожа на шаблон MVC, але побудована таким чином, щоб підвищити повторне використання коду.

— повторне використання. Всі компоненти є незалежними та інкапсульованими. Таким чином фреймворк надає можливість використовувати ті самі компоненти у декількох місцях програми (залежності можуть надаватись за допомогою DI-контейнеру).

— читабельність. Інкапсуляція покращує читабельність коду, допомагає новим розробникам адаптуватись та швидко орієнтуватись в коді.

— тестування. Всі компоненти є незалежними, це спрощує написання юніт-тестів.

— підтримка готових систем. Компоненти інтегруються, вони можуть бути легко замінені на інші при необхідності. Надає можливість швидкого оновлення коду розробниками [9].

3.4 Опис середовищ розробки

Для створення серверної частини було використано інтегроване середовище розробки (IDE) Microsoft Visual Studio 2017, розроблену та підтримувану корпорацією Microsoft.

Visual Studio – це інструмент, який використовується для написання, підтримки, відладки та публікації написаного коду. Інструментарій VS також включає компілятори, різні графічні інструменти (для створення UML – діаграм, для підходу EntityFramework Model First, тощо) які значно полегшують роботу над створенням програмних засобів та їх підтримкою. [10].

Функції VS, які пришвидшують роботу над розробкою:

Хвилясті лінії внизу написаного рядка коду надають інформацію про можливі проблеми або помилки у коді при розробці. Ця інформація є візуальною та з’являється на екрані в реальному часі під час написання, що дозволяє помічати та виправляти помилки відразу, до етапу збірки програми. При наведенні курсору на підкреслений текст буде показано деталі попередження. “Швидкі дії” представлені жовтою лампою надає функціонал для редагування коду, рефакторингу. Також, у разі виконання на полі з помилкою, надається список можливих операцій, які можна застосувати для усунення проблеми

Інструментарій для рефакторингу надає можливості для інтелектуального перейменування різних елементів (змінні, методи, класи і т.д.), зміна сигнатури методів, зміна структури інтерфейсів та класів та ін. [10].

Для створення клієнтської частини було використано IDE WebStorm, розроблену та підтримувану корпорацією JetBrains.

WebStorm – потужна IDE для сучасної JavaScript-розробки, обладнана для комплексного розвитку на стороні клієнта і розробки на серверній стороні використовуючи Node.js [11].

WebStorm забезпечує розширену підтримку JavaScript, HTML, CSS та їх сучасних альтернатив, а також таких структур, як Angular або React. WebStorm також інтегрується з різними інструментами веб-розробки і системами контролю версій.

Переваги використання WebStorm:

— середовище розробки забезпечує підтримку наступних засобів: JavaScript, Node.js, ECMAScript 6, TypeScript, CoffeeScript і Dart, а також для HTML, CSS, Less, Sass і Stylus.

— допомога при розробці – підсвічування синтаксису, дослідження документації та рефакторинг.

— аналіз коду на льоту, виділення помилок та швидкі виправлення.

— передова система рефакторингу;

— забезпечення навігації по створюваній системі.

— підтримка сучасних структур: React, Angular, AngularJS, Vue.js, Express та інше.

— інтегрований відладник для коду на стороні клієнта та Node.js. [11].

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ

Структура програмного продукту складається з серверної частини, реалізованої на платформі .NET, використовуючи ASP.NET WebAPI 2 для взаємодії з клієнтом, що являє собою API з використанням архітектурного стилю REST, та клієнтської частини на Angular.

4.1 Архітектура системи

Систему реалізовано за клієнт-серверною архітектурою (рисунок 4.1).

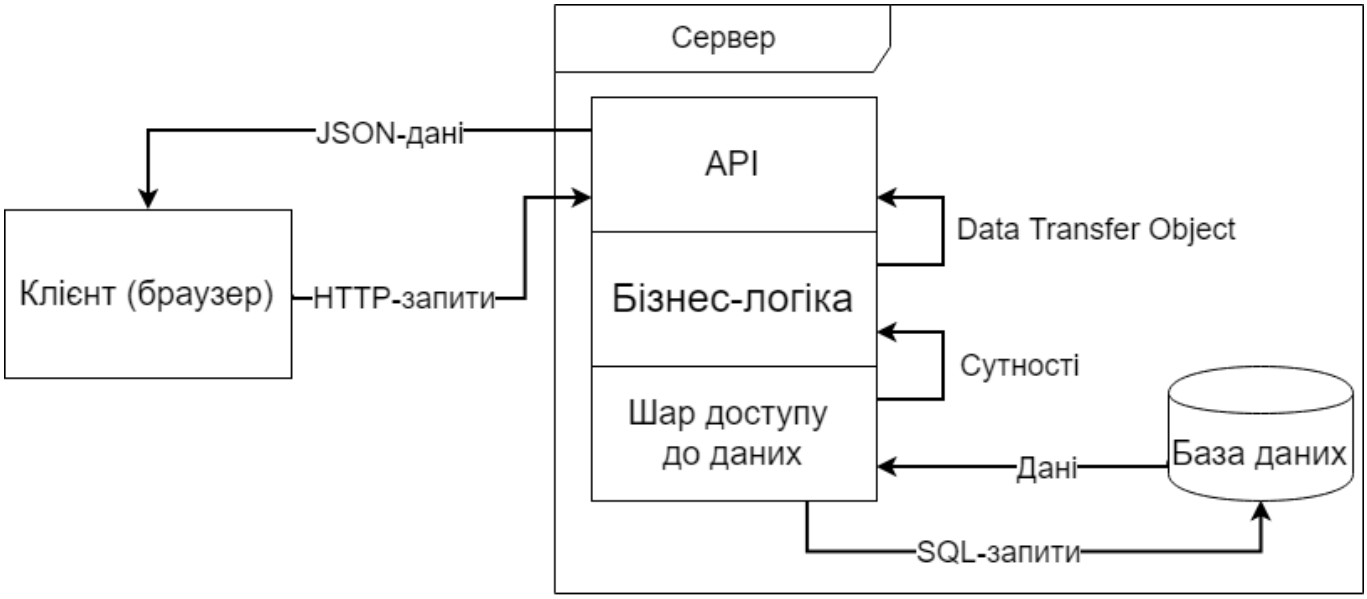

Рисунок 4.1 – Архітектура системи

Серверна частина реалізована за 3-рівневою архітектурою та складається з проектів доступу до бази даних (DAL (Data access layer)), бізнес-логіки (BL (Business layer)) та рівня представлення, котрий представляє собою API для роботи з клієнтом [12]. Рівень представлення реалізовано, використовуючи архітектурний стиль REST, використовуючи такі HTTP методи як GET, POST, PUT та DELETE (рисунок 4.2) [13].

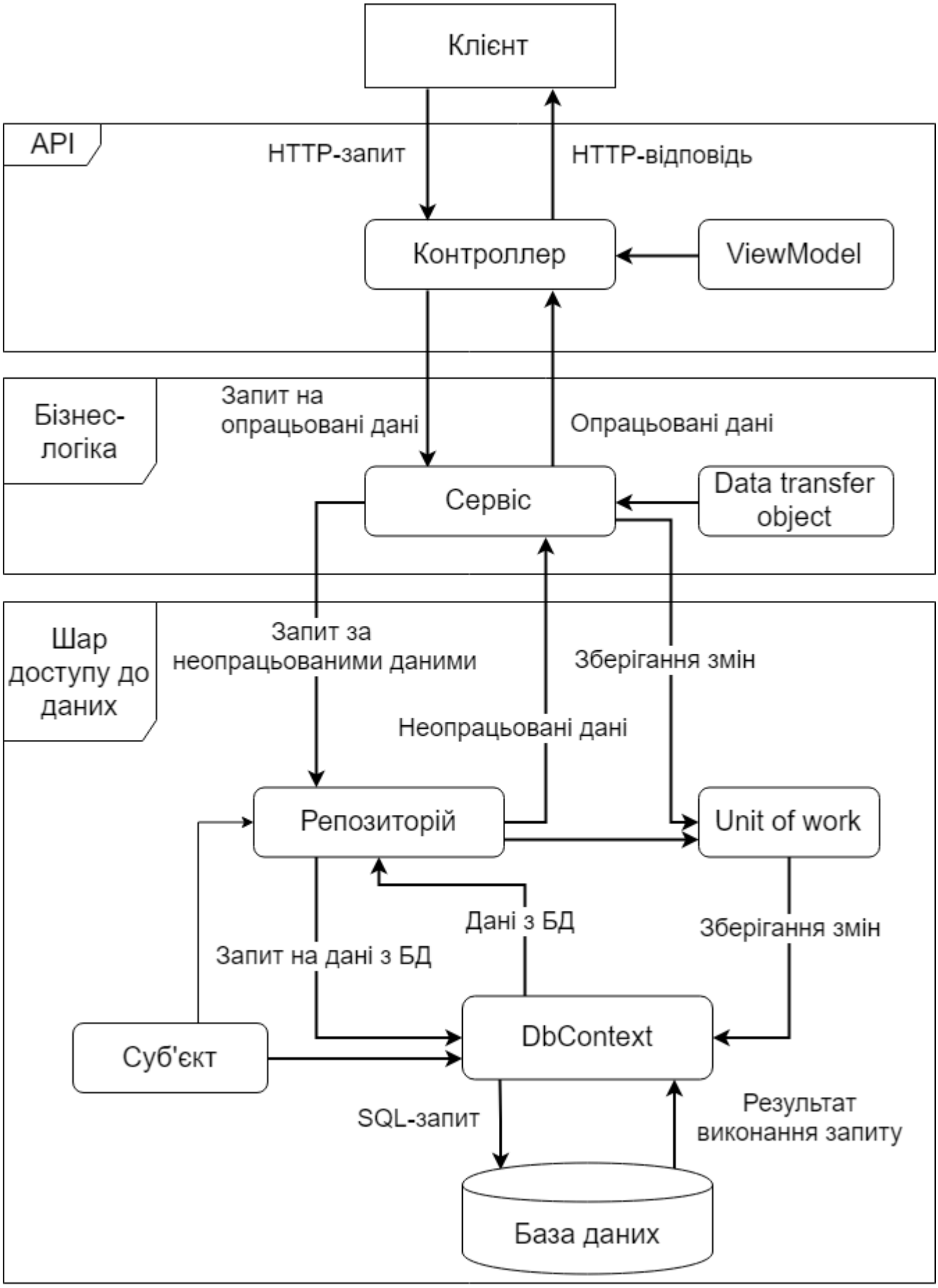

Рисунок 4.2 – Архітектура серверної частини

При отриманні HTTP запиту від клієнтської частини у відповідності за URL за допомогою системи маршрутизації створюється екземпляр контроллеру та викликається відповідний метод опрацювання запиту. Дані, отримані від клієнту, конвертуються та, в разі необхідності, десеріалізуються до відповідних моделей контроллера. Проходить їх перевірка та для подальшого опрацювання запиту використовується відповідний сервіс для опрацювання логіки запиту. В разі наявності даних моделі вона конвертується до відповідної моделі сервісу (DTO).

Сервіси містять необхідні засоби для обробки даних, отриманих від клієнта чи отримання та опрацювання даних з БД. Для доступу до даних з БД сервіс використовує відповідні репозиторії. Для доступу до них використовується паттерн unit of work (UoW).

Паттерн UoW надає можливість абстрагуватись від класу контексту та надати функціонал оперування репозиторіями та збереження внесених в БД змін (аналог DbContext.Save) [14]. Вони використовують клас ApplicationContext, що наслідується від класу DbContext та є ORM-представленням схеми концептуальної моделі, що використовується для відображення класів системи на модель бази даних [15].

Для передачі інформації між API – сервером та клієнтом використовується формат даних JSON. Необхідні дані серіалізуються до JSON – формату даних на сервері при формуванні HTTP – відповіді для GET методу, та десеріалізуються за відповідною структурою на клієнті. Для HTTP – методів POST та PUT дані, які необхідно відправити, на клієнтській частині заповняються в формі. При відправленні проходить валідація, і, якщо вона успішна, дані з цієї форми конвертуються у відповідну модель клієнта, серіалізуються до JSON – формату, та надсилаються на сервер за відповідним методом. На сервері дані десеріалізуються та проходить їх повторна валідація.

Клієнтську частину реалізовано за допомогою TypeScript-фреймворка Angular. Інструментом відображення виступають Angular-компоненти, для зв’язку з сервером використовуються сервіси [9].

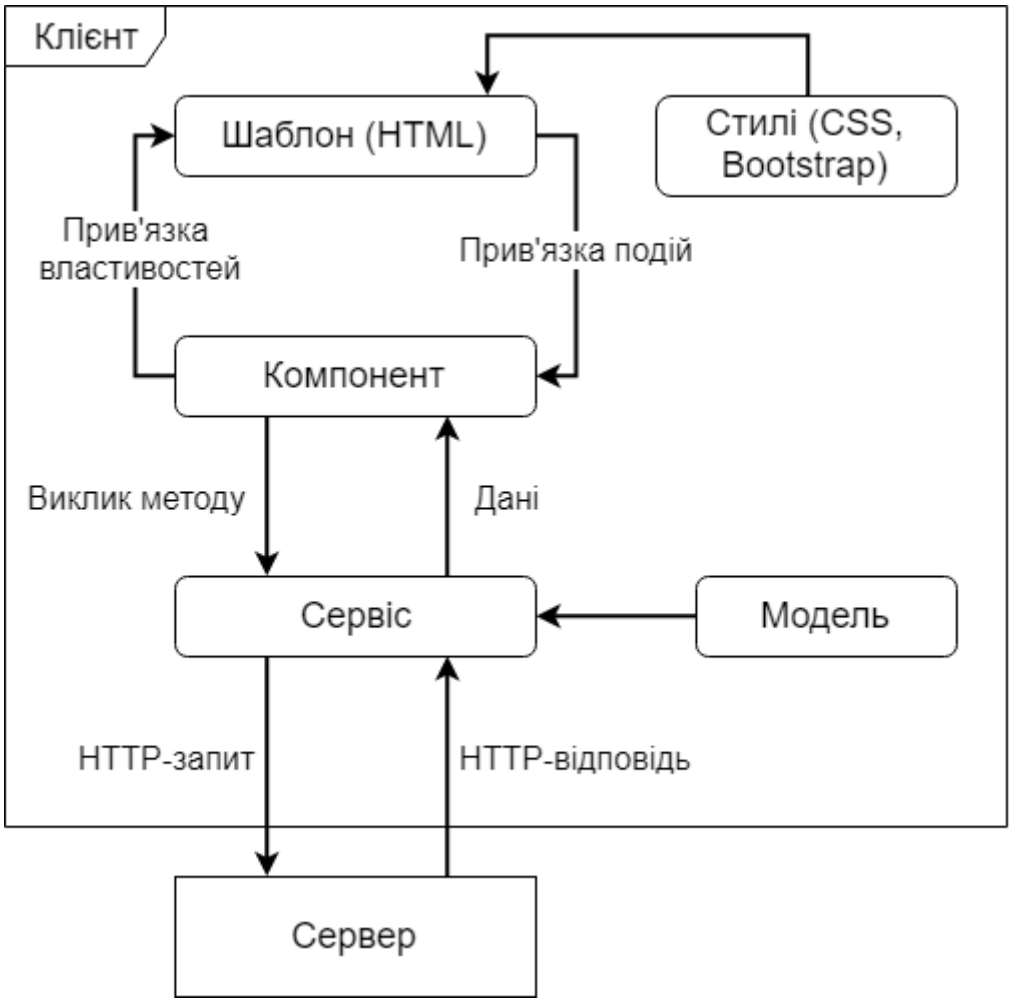

Рисунок 4.3 – Архітектура клієнтської частини

Користувацький інтерфейс реалізовано за допомогою компонентів. Шаблон використовує властивості, які містяться у відповідному компоненті. Таким чином реалізовано прив’язку даних, отриманих з сервера з відповідними елементами шаблону. Для забезпечення інтуїтивності інтерфейсу необхідно гарантувати користувачу миттєвий відгук на його дії. Для цього при створенні шаблону передбачено прив’язку подій, які можуть виникнути при діях користувача до відповідних методів компоненту або переходу до іншого компоненту [16].

Для стилізації кожний компонент містить окремий .css-файл та глобальний файл. Також використовується Bootstrap [16].

Для відправлення запитів на сервер створено сервіси. Вони відправляють запити за маршрутами, вказаними в контроллерах серверної частини. При виклику метода сервісу як параметр передаються дані для запиту [17]. Якщо дані мають містити модель (для POST та PUT методів), вона передається через заповнені дані

форми. Сервіси надають функціональні можливості, що повністю покриває функціональні можливості та маршрути контроллера.

Для полегшення роботи з даними, які використовуються клієнтом, було створено власну структуру моделей для клієнта, структура якої була адаптована для полегшення роботи зі зв'язними даними на клієнтській частині. При відправці моделей або при отриманні відповіді з сервера вони серіалізуються в формат даних JSON або десеріалізуються з нього в тип зазначених моделей. Це забезпечує полегшення роботи з даними та прив'язки даних до елементів представлення.

Користувач працює тільки з клієнтом, котрий відправляє HTTP–запити до API серверу, для передачі даних до клієнта сервер використовує SQL–запити для отримання необхідної інформації з бази даних.

4.2 Опис функціональних можливостей системи

Основні функціональні можливості системи та їх поділ між користувачами можна описати за допомогою діаграми прецедентів

Рисунок 4.4 – Частина діаграми прецедентів для ролі адміністратора

Адміністратору надано функціонал для оперування обліковими записами користувачів та оперування ролями облікових записів. Система повинна бути закритою для змін для простого користувача, неможливо пройти реєстрацію та отримати доступ до даних для будь-якого користувача. Створення облікових записів проводиться лише адміністратором

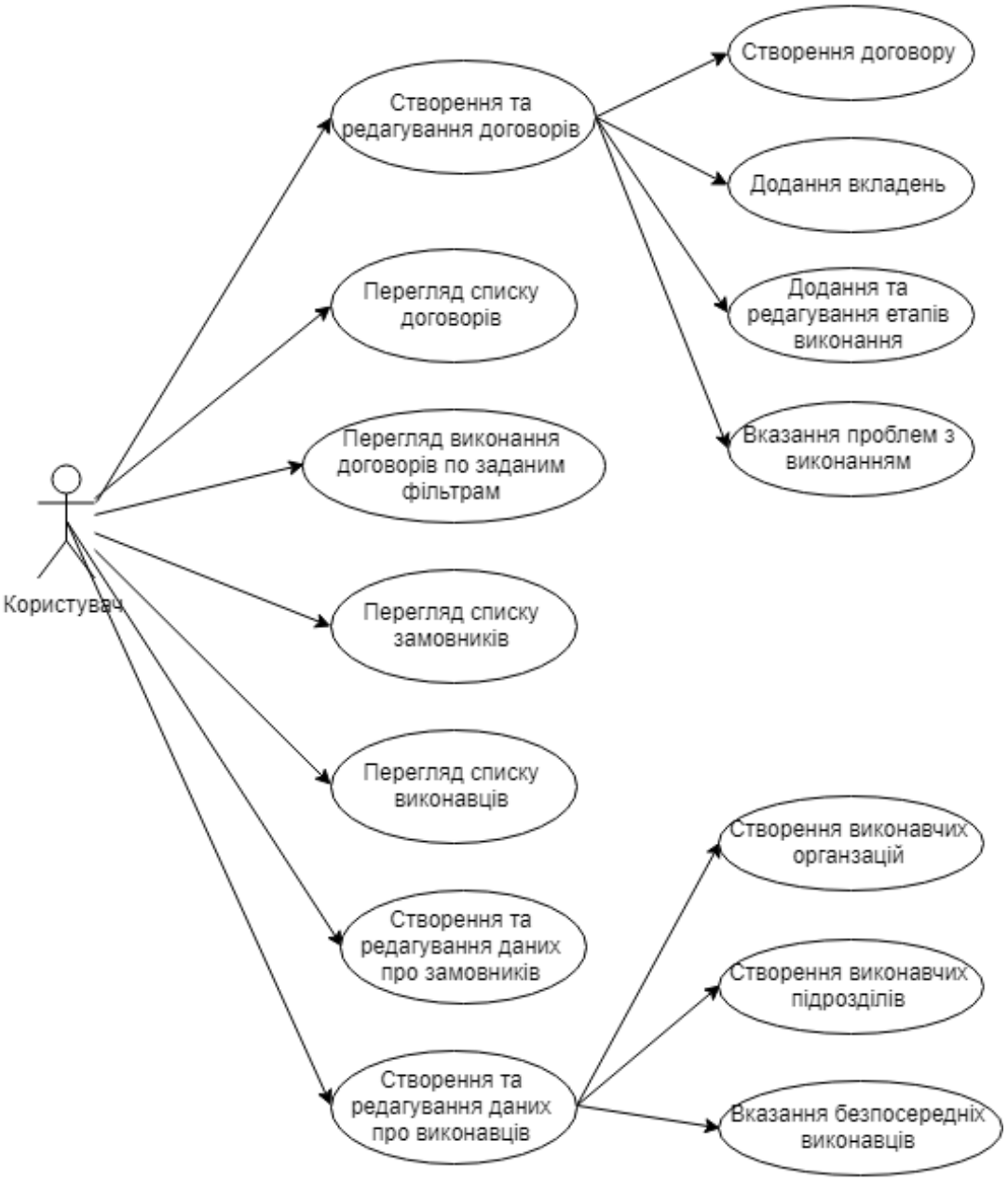

Рисунок 4.5 – Частина діаграми прецедентів для ролі користувача

Користувач – основна роль в системі. Користувачі ведуть облік даних, створюють, редагують дані в системі, мають можливість перегляду та аналізу введених даних. Обліковий запис користувача має бути створений адміністратором системи.

4.3 Діаграма класів

На рисунках 4.6-4.7 наведено діаграми класів.

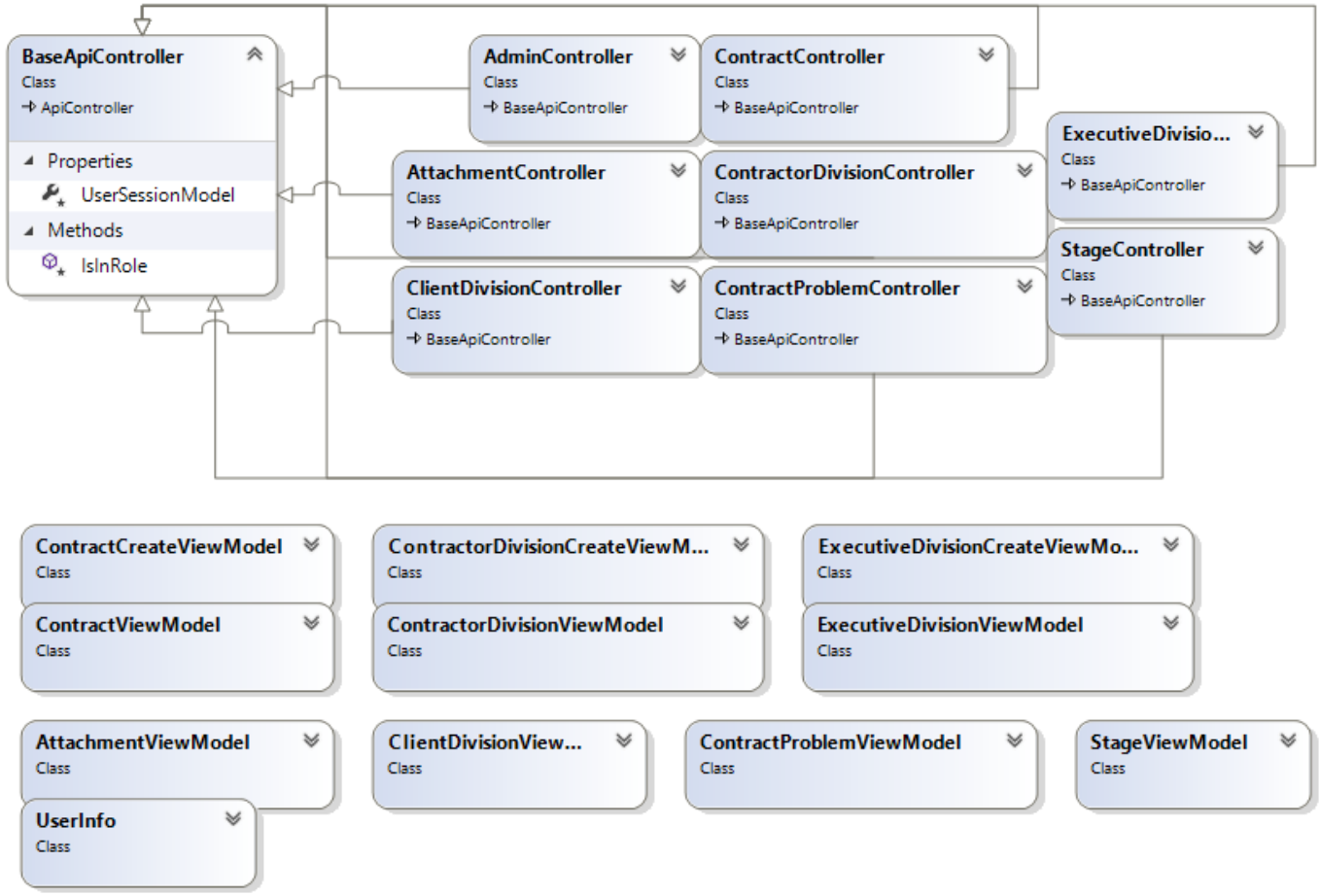

Рисунок 4.6 – Діаграма класів API (веб-додатку)

Обробка HTTP-запитів проводиться контроллерами. За URL створюється необхідний екземпляр контроллера. Створені контроллери наслідується від класу BaseApiController, котрий надає функціонал для роботи над процесом авторизації з підтримкою ролей та наслідується від класу ApiController, котрий є базовим контроллером WebAPI та надає базовий функціонал для обробки запиту [18].

Сервіс використовує моделі даних, що називаються ViewModel [19]. Це надає змогу спростити роботу API з клієнтом та складними моделями та надавати користувачу лише ту інформацію, яка потрібна в відповідній ситуації.

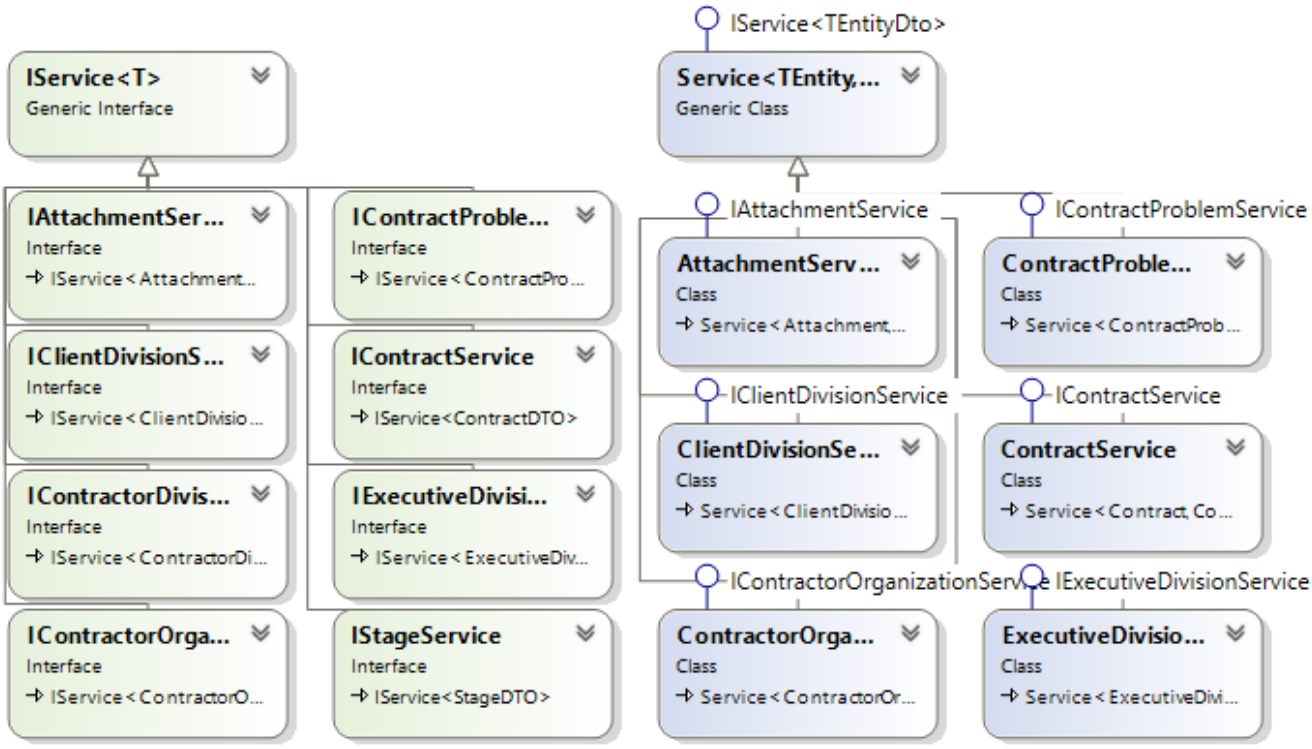

Рисунок 4.6 – Діаграма класів BLL (бізнес-логіки)

Далі обробка запиту проходить шаром бізнес логіки. Викликається відповідний метод на екземплярі класу сервісу, що надає функціонал для роботи з даними та наслідується від базового абстрактного класу сервісу Service<TEntity, TEntityDTO> (generic), що реалізує інтерфейс IService<T> (generic) та надає базові методи роботи з даними. Сервіс використовує моделі даних, що називаються DTO (Data Transfer Object) [21]. Це розмежування дає змогу змінювати, додавати та спрощувати моделі, створені в шарі доступу до даних.

Для роботи з базою даних сервіс використовує відповідний екземпляр класу репозиторію, котрий надає функціонал роботи з БД, наслідується від базового абстрактного класу Repository<T>, котрий реалізує інтерфейс IRepository<T>. Для представлення бази даних в системі реалізовано клас UnitOfWork, котрий надає екземпляри репозиторіїв для відповідних моделей бази даних та збереження внесених в неї змін. Цей клас дозволяє абстрагуватись від роботи з класом, що наслідується від DbContext [14].

Репозиторій працює з класом ApplicationContext, що наслідується від DbContext і є ORM-представленням бази даних, його властивості (класу DbSet<T>) представляють таблицю в БД.

Створення БД відбувається автоматично, якщо вона не існує. Для цього використовується клас контексту ApplicationContext, концептуальну модель БД, що представлено у вигляді класів, та конфігурація полів класів та залежностей, які мають бути встановлені в БД та зв'язність даних при роботі з контекстом. Для конфігурації використано FluentAPI [23].

Створену архітектуру класів було адаптовано для підтримки DI (Dependency injection)-контейнерів. В даному випадку використано Autofac. Це надає змогу вилучити роботу над необхідними залежностями і перенести цю роботу на DI-контейнер [24].

Для підтримки ролей облікових записів в системі було створено власну систему авторизації. Авторизація здійснюється на основі токенів. Клієнт надсилає HTTP-запит з "grant_type", що містить дані облікового запису, що необхідні для авторизації – email та пароль. На основі цих даних проводиться пошук облікового запису в системі, якщо користувача за внесеними даними знайдено - генерується токен доступу та повертається у відповідь клієнтові. Таким чином можна визначити однозначно визначити ролі поточного користувача в системі. Для всіх операцій з системою необхідно авторизуватись, неавторизованому користувачеві (або користувачаві з недійсним токеном) при спробі відправити запит сервером буде повернуто 401 Unauthorized, клієнт не допускає до роботи з системою неавторизованого користувача [22]

Для реалізації системи було використано OWIN, створено власні провайдери для валідації користувача та задано ролі в системі. Реалізовано функціонал для обробки запиту для перевірки авторизації та виділення ролей користувача. [20].

4.4 Модель бази даних

Діаграма класів, що представляє модель БД представлена на рисунку 4.7

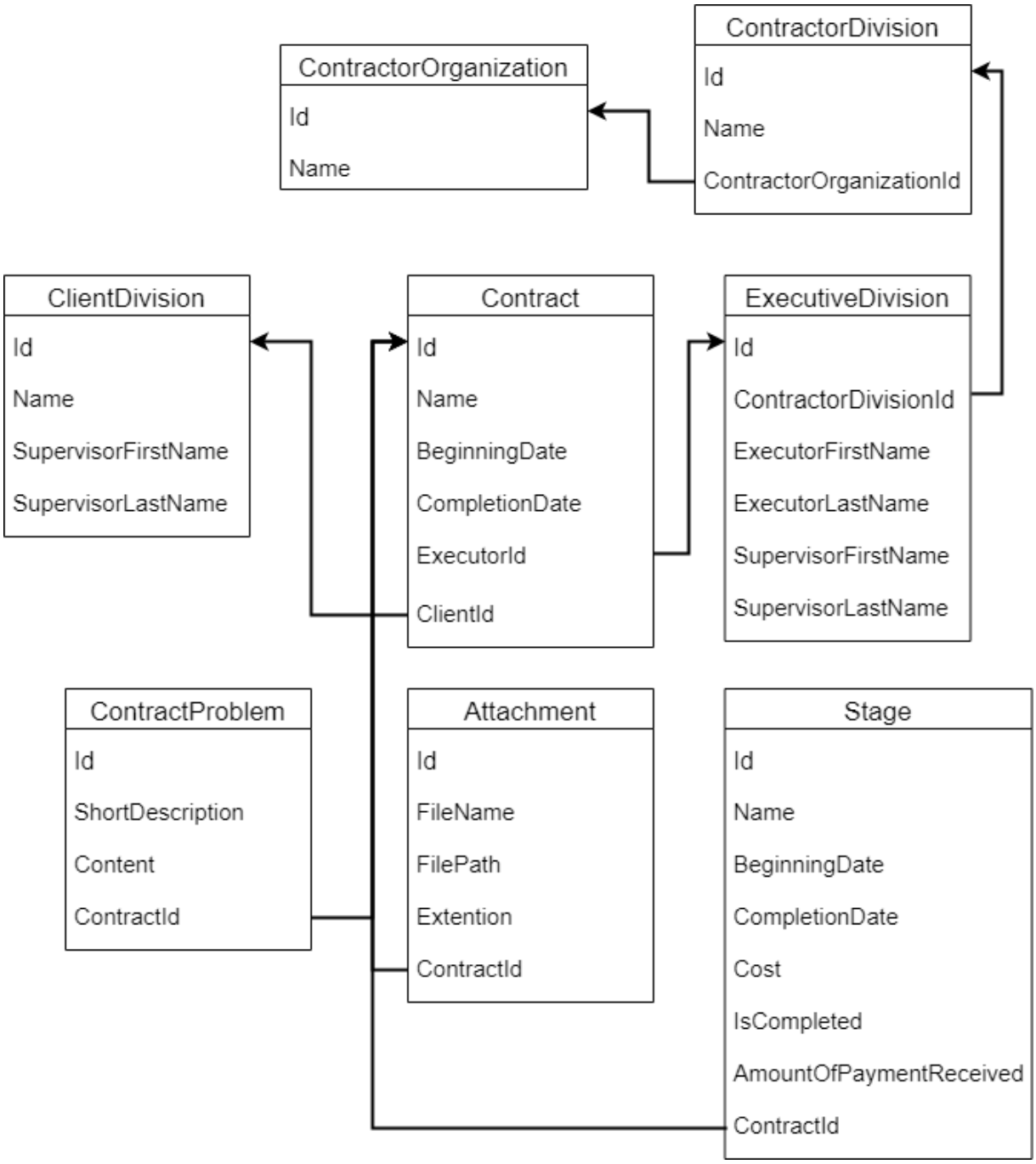

Рисунок 4.7 – Діаграма класів

Головною частиною діаграми є клас договору. Навколо взаємодії з договором побудована решта класів. Є можливість додання додатків, вони зберігаються окремо від бази даних за попередньо вказаним шляхом

4.5 Опис таблиць бази даних

На таблицях 4.1 – 4.8 надано структури таблиць бази даних

Таблиця 4.1 Структура таблиці “Організація виконавця”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
Name	nvarchar(450)	Назва організації

Таблиця 4.2 Структура таблиці “Організація замовника ”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
Name	nvarchar(450)	Назва організації
SupervisorFirstName	nvarchar(450)	Ім’я відповідального
SupervisorLastName	nvarchar(450)	Прізвище відповідального

Таблиця 4.3 Структура таблиці “Підрозділ виконавця”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
Name	nvarchar(450)	Назва підрозділу
ContractorOrganizationId	bigint	Ім’я виконавця

Таблиця 4.4 Структура таблиці “Вкладення”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
FileName	nvarchar(450)	Ім’я файлу
FilePath	nvarchar(450)	Шлях, за яким збережено файл
Extention	nvarchar(450)	Розширення
ContractId	bigint	Id договору

Таблиця 4.5 Структура таблиці “Проблема з договором”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ

Продовження таблиці 4.5 “Проблема з договором”

ShortDescription	nvarchar(450)	Короткий опис проблеми
Content	nvarchar(450)	Id користувача
ContractId	bigint	Id договору

Таблиця 4.6 Структура таблиці “Виконавець”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
ContractorDivisionId	bigint	Назва набору камер
ExecutorFirstName	nvarchar(450)	Id користувача
ExecutorLastName	nvarchar(450)	Id користувача
SupervisorFirstName	nvarchar(450)	Id користувача
SupervisorLastName	nvarchar(450)	Id користувача

Таблиця 4.7 Структура таблиці “Договір”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
Name	nvarchar(100)	Тема договору
BeginningDate	Date	Дата початку роботи над етапом
CompletionDate	Date	Дата планованого закінчення роботи над етапом
ExecutorId	bigint	Id виконавця
ClientId	bigint	Id замовника

Таблиця 4.8 Структура таблиці “Етап”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
Id	bigint	Первинний ключ
Name	nvarchar(100)	Назва етапу

Продовження таблиці 4.8 “Етап”

BeginningDate	Date	Дата початку роботи над етапом
CompletionDate	Date	Дата планованого закінчення роботи над етапом
IsCompleted	bigint	Чи є етап завершеним
Cost	money	Вартість етапу
AmountOfPaymentReceived	money	Фактична оплата
ContractId	bigint	Id договору

Для створення бази даних було використано підхід Entity Framework Code First. Для цього в проекті доступу до даних було створено класи доменних моделей які представляють кожну з таблиць бази даних [25]. Для доступу до даних, що зберігаються в БД використовується клас DbContext, generic-клас DbSet представляє таблицю БД. Всі відношення реляційних БД конфігуруються в окремих файлах за допомогою FluentAPI [23]. Робота з базою даних здійснюється за допомогою ORM-системи Entity Framework [15].

5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ НАУКОВО-ДОСЛІДНИХ РОБІТ

Розроблена програмний комплекс розроблений з використанням веб-технологій і тому працює в браузерах, які підтримують актуальні веб-стандарти. Клієнтська частина використовує компоненти Angular для відображення елементів шаблону, тому існує прив’язка даних між шаблоном та моделями компонента. Таким чином, при зміні моделі компонента відповідна частина шаблону змінюється автоматично, тому дані завжди є актуальними та не потребують оновлення сторінки після виконання будь-яких змін. Також реалізовано систему автоматичної переадресації для зручності роботи з системою.

5.1 Інсталяція та системні вимоги

Система складається з двох частин. Для використання клієнтської частини користувачу потрібний лише браузер та стабільне інтернет-з’єднання.

Для надання можливості роботи з системою необхідно розмістити серверну та клієнтську частину систему на сервері.

5.2 Інструкція з використання програмного продукту

Для початку користування системою необхідно авторизуватися. Неавторизованому користувачу не надано прав перегляду або зміни даних в системі. Будь-які спроби здійснити подібні дії будуть відхилені системою, також буде здійснено перехід до компоненту авторизації. Відповідний компонент зображено на рисунку 5.1. Для авторизації необхідно ввести Email та пароль, задані адміністратором при створенні облікового запису.

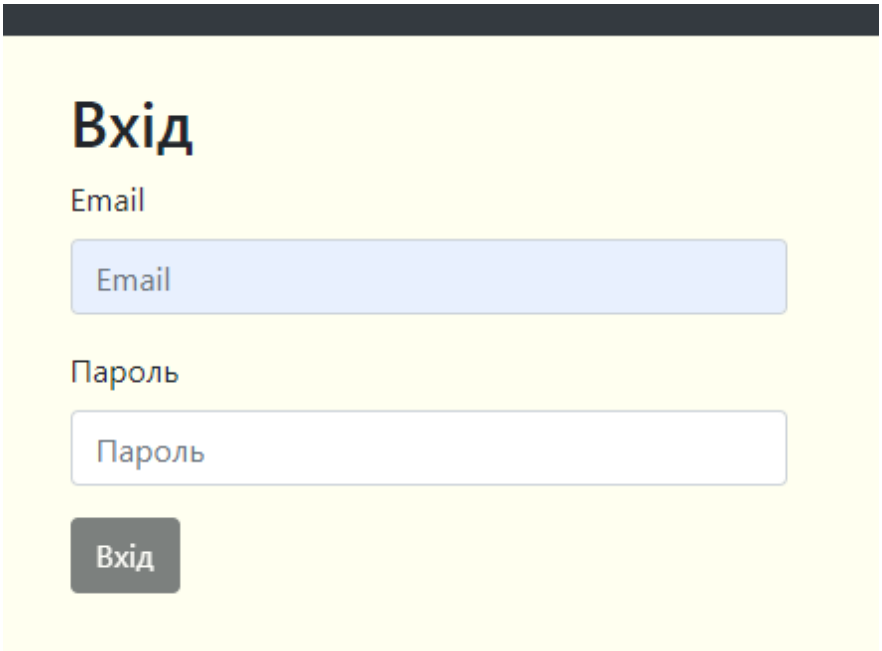

Рисунок 5.1 –Форма авторизації

Реалізовано просту перевірку введених значень, якщо вона не проходить, кнопка submit (“Вхід”) не є доступною для натискання. Результатом виконання запиту на сервері є Bearer – жетон, який є унікальним для користувача та дозволяє за ним на сервері однозначно визначити користувача, що відправив запит (рисунок 5.2).

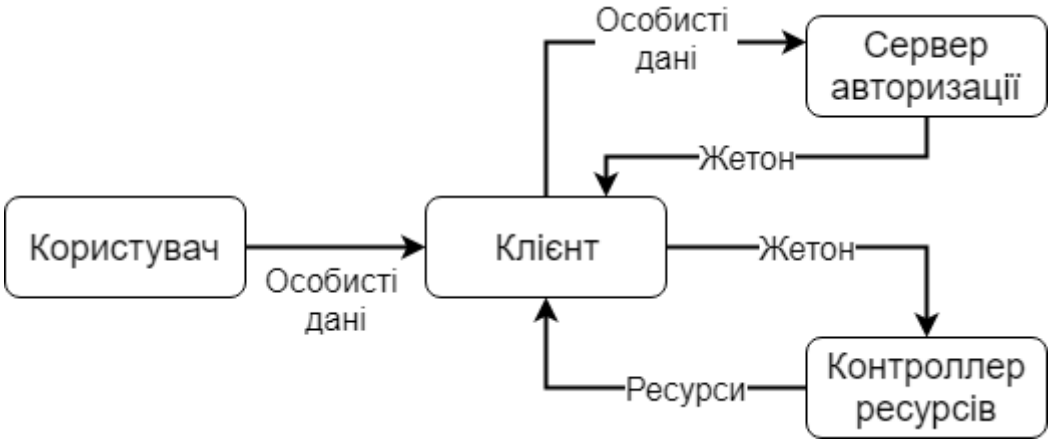

Рисунок 5.2 – Схема авторизації

Якщо користувач авторизований, в нього є доступ до навігаційного меню, яке надає можливість швидко здійснювати навігацію по основним функціям системи. Є можливість переглянути список договорів, списки виконавців, замовників, виконавчих підрозділів та організацій та переходу до компоненту відображення даних для перегляду виконання договорів. Також доступна можливість виходу з облікового запису (рисунок 5.3). Натискання на кнопки навігаційного меню викликає

автоматичний перехід до відповідних списків даних (за виключення кнопки виходу з облікового запису). Після натискання на кнопку виходу збережений токен авторизації та всі дані про користувача видаляються з клієнта, для доступу до системи необхідно повторно авторизуватись для отримання нового токена, буде автоматично здійснено перехід до компоненту авторизації.

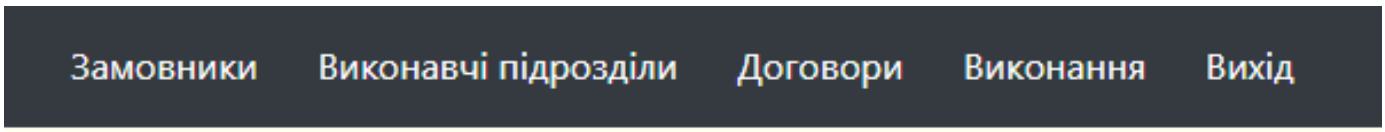

Рисунок 5.3 – Навігаційне меню

Якщо авторизація пройшла успішно, буде здійснено перехід до сторінки зі списком договорів. Відповідний компонент зображено на рисунку 5.4.

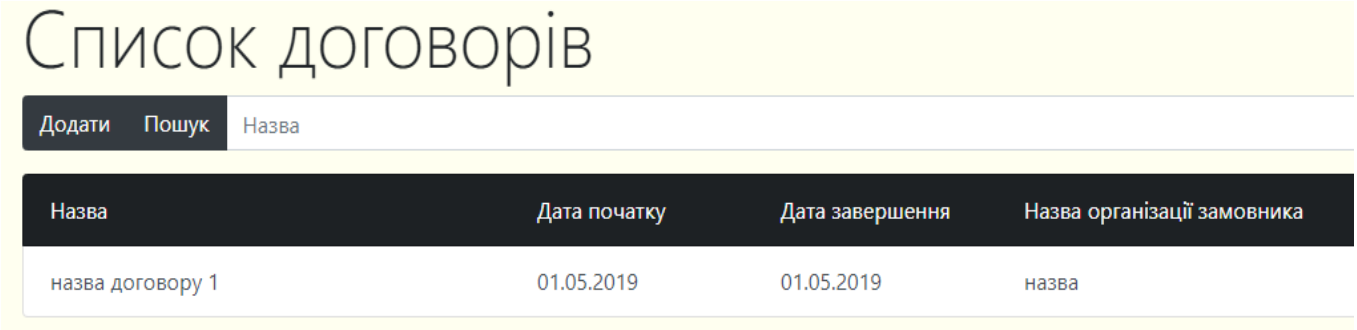

Рисунок 5.4 – Список договорів

В списку договорів доступна базова інформація для перегляду та навігації за договорами – назва договору, дата початку та його планована дата завершення та назва організації замовника. Доступний функціонал для фільтрації списку договорів, та додавання нового договору. Для перегляду деталей договору потрібно натиснути на відповідний договір в списку.

Для переходу до компоненту перегляду списку замовників або виконавчих підрозділів потрібно натиснути на відповідну кнопку навігаційного меню. Для кожного елементу списку доступна наступна інформація – назва організації, прізвище та ім’я відповідальної за договір особи для списку замовників та назва організації виконавця та підрозділу виконавця для виконавчих підрозділів. Доступний функціонал для фільтрації списку за доступними за списком полями та додання елементів за натисканням відповідної кнопки. Видалення елементів відбувається

шляхом натискання кнопки “Видалити” у елементу списку, котрий необхідно видалити ().

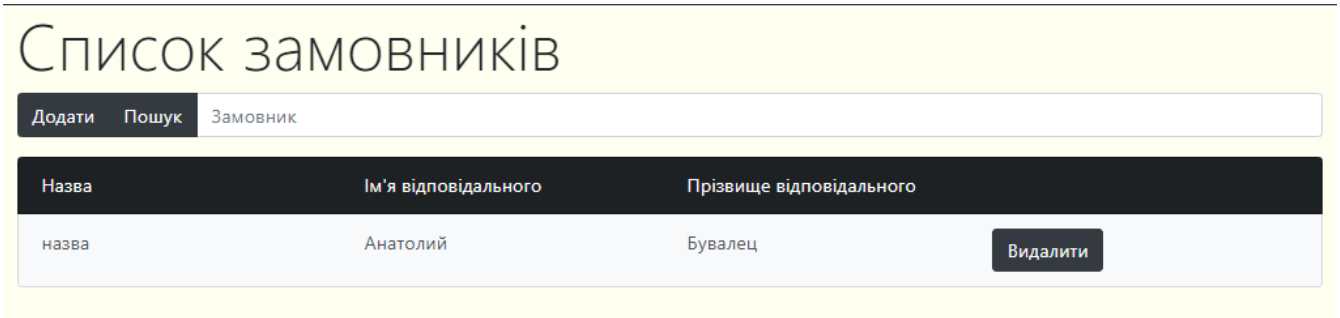

Рисунок 5.5 – Список замовників

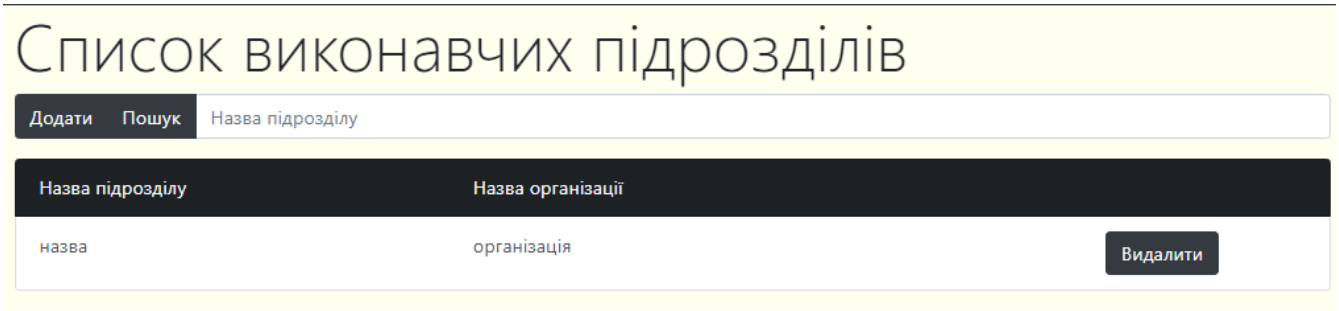

Рисунок 5.6 – Список виконавчих підрозділів

Для переходу до редагування або перегляду елементів потрібно натиснути на необхідний елемент списку і буде здійснено автоматичний перехід до необхідного компоненту.

Для доступу до необхідних записів у списках можна скористатись пошуковим рядком. Для договорів доступна система фільтрів по таким категоріям як:

— назва організації замовника;

— назва організації виконавця;

— назва підрозділу виконавця;

— проміжок часу;

— імена учасників.

Для додавання, перегляду або зміни моделей в списках передбачено окремі компоненти, які можна викликати за натисканням відповідних кнопок. Компонент додавання моделі можна викликати при натисканні кнопки “Додати” у відповідних списках (рисунок 5.7).

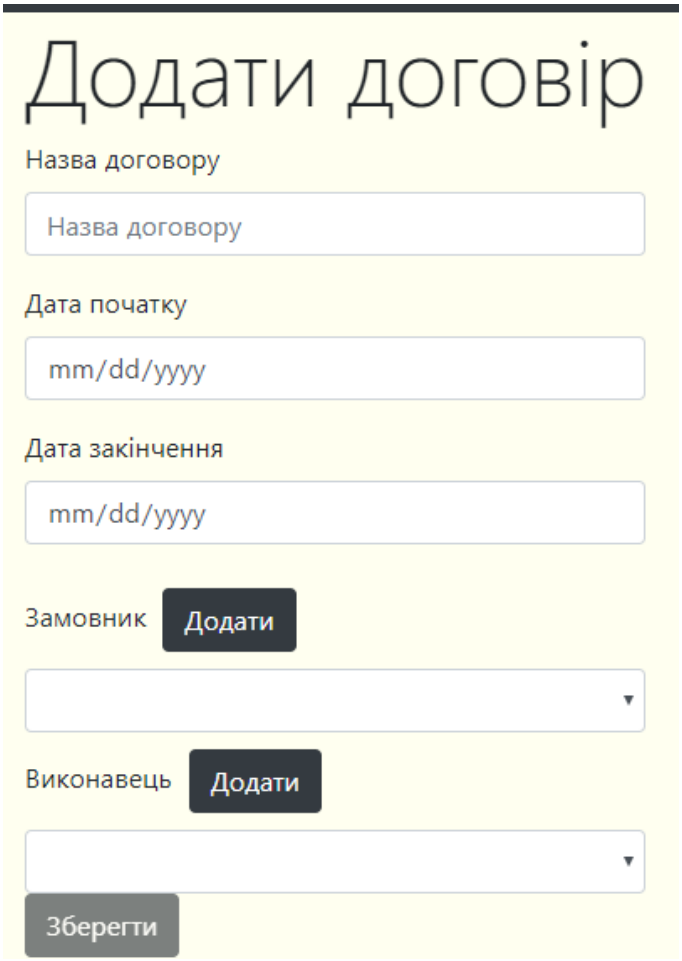

Рисунок 5.5 – Компонент додавання договору

Для додавання договору необхідно обрати існуючу організацію замовника та виконавця. Компонент додавання договору містить опцію для переходу до компоненту створення замовника або виконавця, якщо це є необхідним (кнопка “Додати” поряд з відповідною опцією).

При спробі створення виконавця необхідно вказати виконавчий підрозділ з випадаючого списку доступних підрозділів. Є можливість переходу до компоненту для додавання виконавчого підрозділу.

Якщо дані, що внесені в форму відправки, не проходять валідацію, то кнопка для відправки даних на сервер не є доступною. Дані замовників та виконавців для додавання договору та дані виконавчих підрозділів для додавання виконавців прив’язуються до форми автоматично та вибір їх користувачем на формі фіксується через їх ідентифікаційний номер. Це гарантує правильне відправлення та опрацювання складних зв’язаних даних.

Після успішного додавання моделі, користувачу надано можливість переглянути її в відповідному списку, переглянути її деталі та редагувати її при натисканні на відповідний елемент списку (рисунок 5.8).

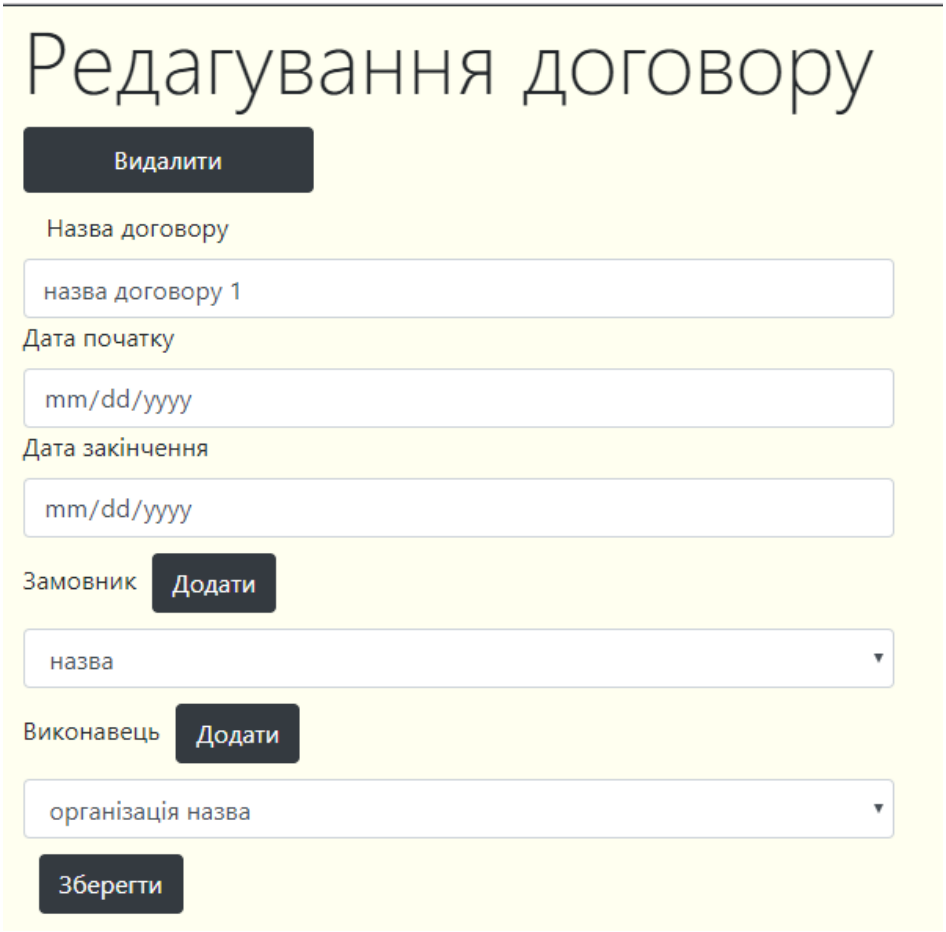

Рисунок 5.8 – Редагування договору

Для редагування моделі реалізовано прив’язку даних вже існуючої моделі, які потрібно редагувати та елементів форми, тому дані полів форми автоматично заповнюються вже існуючими даними моделі. Також є можливість видалити поточну модель.

Після успішного додавання договору є можливість додати, редагувати або видалити етапи виконання та проблеми, які виникли при роботі з договором, прикріпити, видалити або завантажити всі необхідні вкладення. Функціонал для роботи з етапами, проблемами та вкладеннями знаходиться в меню редагування договору.

Інструментарій для редагування етапів, проблем та вкладень знаходиться нижче форми редагування договору. Їх зміна проводиться окремо від редагування полів договору.

Першою частиною є оперування вкладеннями. В користувача є можливість збереження на сервер існуючого файлу, зв’язаного з договором, шляхом вказання файлу та натискання кнопки “Завантажити”. Після збереження користувачеві надано інструменти для видалення або збереження вкладення. Список автоматично оновлюється при внесенні будь-яких змін. Для завантаження вкладення потрібно натиснути на його елемент списку вкладень (рисунок 5.8).

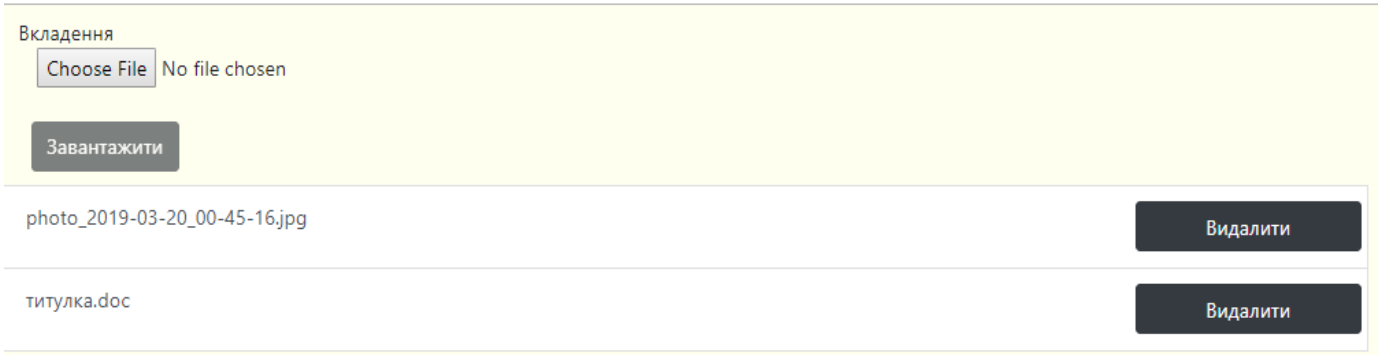

Рисунок 5.8 – Оперування вкладеннями договору

Наступною частиною є оперування проблемами договору. Для додання проблем передбачено окремий компонент. Для відображення проблеми договору використовуються поля короткого опису для обліку та детального пояснення проблеми (рисунок 5.9).

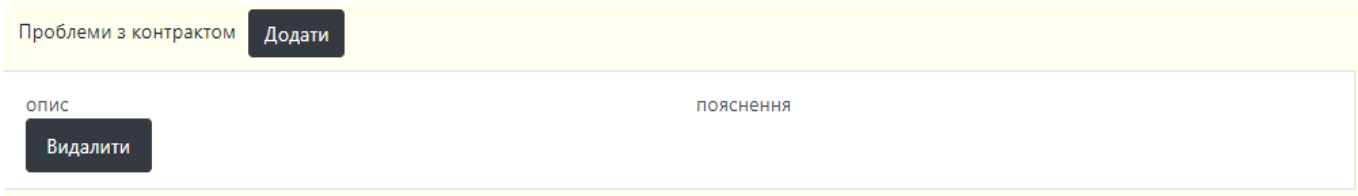

Рисунок 5.9 – Оперування проблемами договору

Останньою частиною редагування договору є редагування етапів виконання договору. Для додавання або редагування етапу передбачено окремий компонент, який викликається натисканням відповідних кнопок (рисунок 5.10).

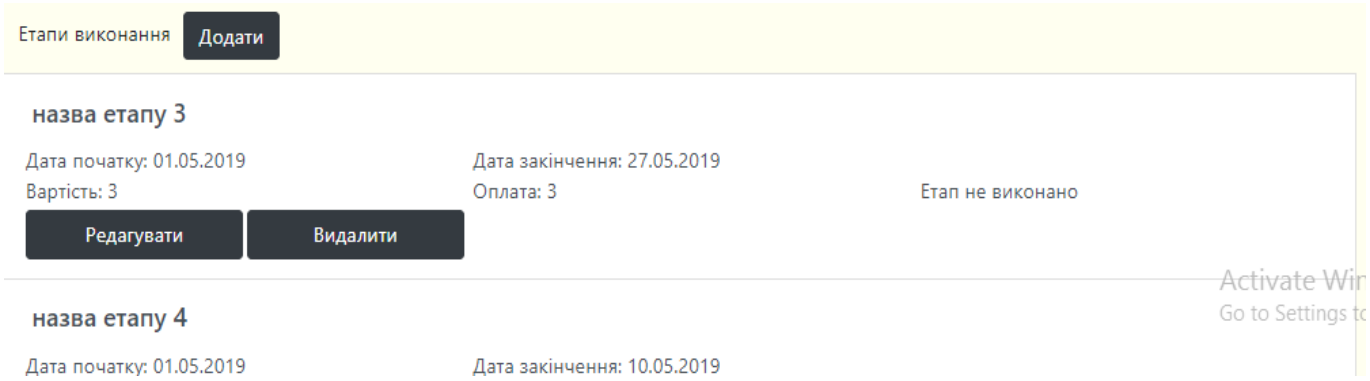

Рисунок 5.10 – Оперування етапами договору

Після завершення редагування договору та зв’язаних з ним моделей та вкладень для збереження змін необхідно натиснути кнопку “Зберегти” для збереження змін, що були внесені в договір. Якщо форма для відправки на сервер не проходить перевірку, кнопка не буде доступною. Після відправки даних, якщо операція пройшла успішно, буде здійснено автоматичну переадресацію на компонент відображення списку договорів.

Щоб перейти в меню перегляду договору потрібно у списку договорів натиснути на рядок з необхідним договором, для редагування вибрати відповідну опцію при перегляді. Видалення договору можливо при перегляді його або при його редагуванні.

Компонент відображення деталей договору відображає всю інформацію, пов’язану з договором: назву, дату початку робіт та їх планованого завершення, виконавці, замовники, проблеми при виконанні, етапи з відображенням їх виконання та прикріплені вкладення (рисунок 5.11). Доступна функція завантаження прикріплених вкладень шляхом натискання на відповідний елемент списку. Завантаження почнеться автоматично.

Для того, щоб перейти до компоненту графічного відображення процесу виконання договорів за їх етапами необхідно обрати відповідний пункт (рисунок 5.12). Доступні наступні критерії для фільтрації договорів, дані про виконання яких необхідно проаналізувати:

— за періодом часу. Основний критерій, необхідний для отримання звітів по виконанню договорів по певному проміжку часу. Результати нанесено на часову

шкалу. Процес виконання договорів та не виконані граничні терміни етапів, статус їх виконання відображається відповідно;

— за організацією замовника, виконавця та безпосередніми виконавцями. Критерій дозволяє проаналізувати дані по виконанню договорів для обраного критерію у табличному вигляді. Процес виконання договорів та не виконані граничні терміни етапів, статус їх виконання відображається відповідно.

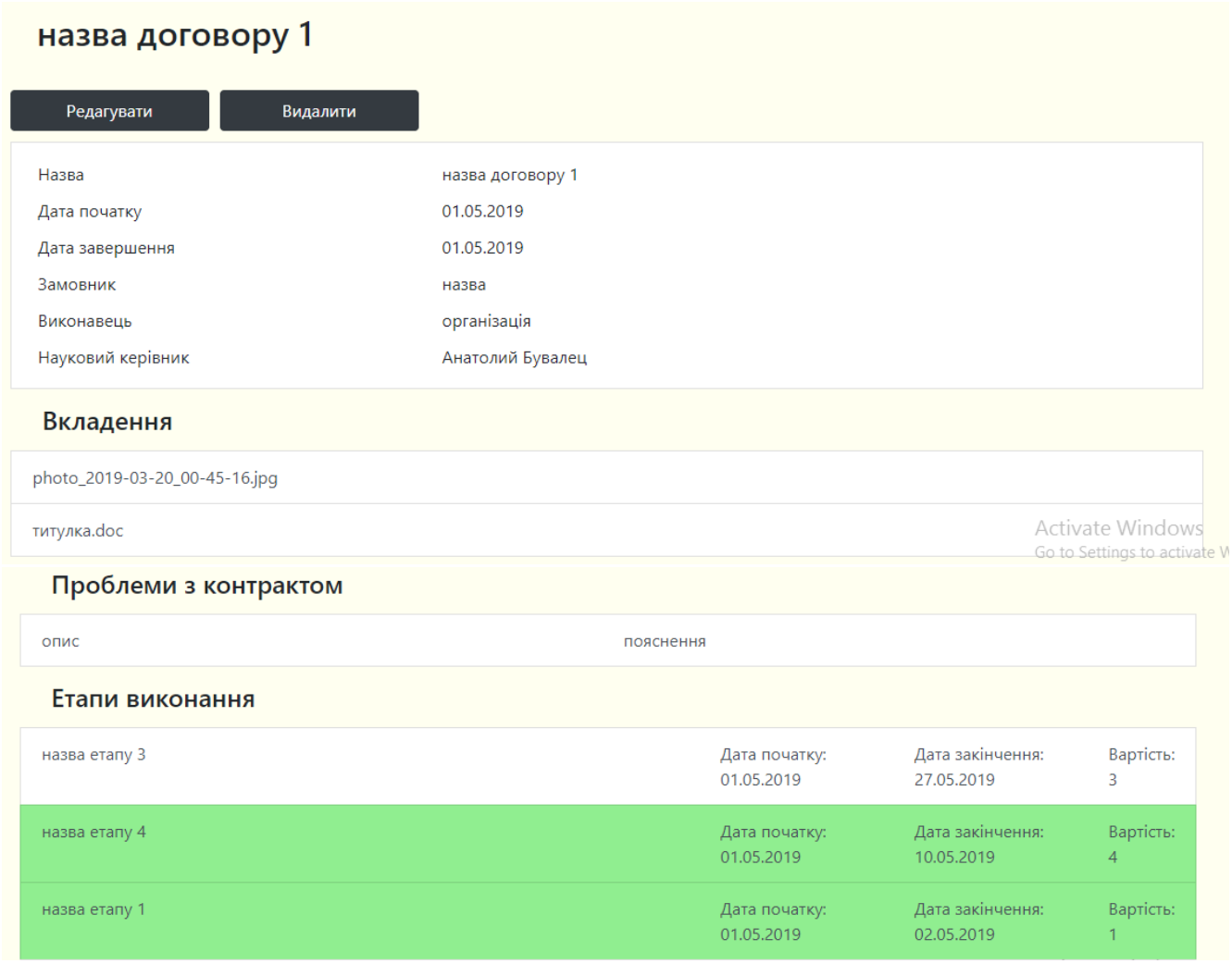

Рисунок 5.11 – Компонент перегляду договору

При відправці форми відбувається фільтрація і дані надаються в табличному вигляді (рисунок 5.12). Для перегляду обрано лише поля назви договору та відсотків їх виконання щоб спростити роботу з даними.

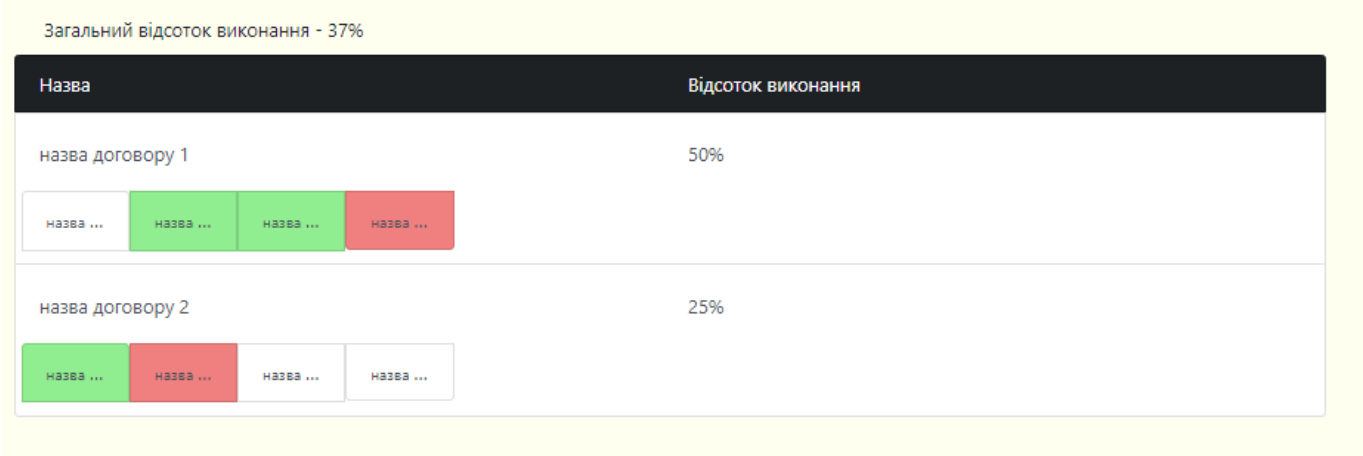

Рисунок 5.12 – Відображення списку договорів з виконанням етапів

Для кожного з договорів вибірки вказано відсоток виконання його договорів. Етапи, які не були виконані вчасно (до вказаної запланованої дати) виділені червоним кольором, вже виконані етапи виділені зеленим кольором. По всій вибірці вказано відсоток виконання етапів.

Після внесення будь-яких змін дані в усіх таблицях та списках автоматично оновлюються, не потрібно оновлювати сторінку вручну. В разі необхідності здійснюється перехід до списку всіх договорів або до останньої відвіданої користувачем сторінки. Для кожного списку передбачено пошуковий рядок для фільтрації і знаходження необхідних даних.

ВИСНОВКИ

Було розроблено програмний продукт для обліку укладених договорів по науково-дослідним роботам. Система дозволяє полегшити роботу з договорами та відслідковуванням процесу їх виконання.

Було виконано дослідження існуючих програмних систем, і зроблено виняток, що вони не задовольняють всім вимогам або не повністю виконують всі необхідні задачі.

Програмний продукт написано за допомогою фреймворків ASP.NET WebAPI мовою C# для серверу та Angular мовою TypeScript для клієнту.

Було реалізовано підтримку авторизації за допомогою ролей, що гарантує захищений доступ до системи використовуючи функціональні можливості адміністратора для створення облікових записів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чичкарев Е. А. Автоматизация учета результатов научной работы сотрудников вуза с использованием системного подхода / Е. А. Чичкарев // Проблеми інформаційних технологій. – 2017. – N 1 (21). – С. 3–5

2. Uyttewaal E. Dynamic Scheduling With Microsoft Project / E. Uyttewaal – Ross Publishing, 2005. – 755 с.

3. Столяров Р. А. Автоматизированная система учета результатов интеллектуальной деятельности в научной организации / А. Р. Столяров // Вопросы территориального развития. – 2015. – № 6 (26) . – С. 4–10

4. Просницкий А. Microsoft Project 2016 Методология и практика / А. Просницкий – LEO Consulting, Киев, 2016. – 407 с.

5. Андерсон Р. Документація Microsoft ASP.NET [Електронний ресурс] / Р. Андерсон. — Режим доступу: https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-2.2

6. Вассон М Документація Microsoft ASP.NET Web API [Електронний ресурс] / М. Вассон — Режим доступу: https://docs.microsoft.com/en-us/aspnet/web-api/

7. Finley K. Microsoft Previews New JavaScript-Like Programming Language TypeScript [Електронний ресурс] / Finley K. — Режим доступу: https://techcrunch.com/2012/10/01/microsoft-previews-new-javascript-like-programming-language-typescript/

8. Vane V. TypeScript Design Patterns / V. Vane – Packt Publishing, 2016. – 256 с.

9. Фримен А. Angular для профессионалов/ А. Фримен – Питер, 2017. – 800 с.

10. Johnson B. Professional Visual Studio 2017 / B. Johnson – John Wiley & Sons, 2017. – 864 с.

11. Hoffmann M. Why I Switched From Visual Studio Code To JetBrains WebStorm [Електронний ресурс] / M. Hoffman — Режим доступу: https://dev.to/mokkapps/why-i-switched-from-visual-studio-code-to-jetbrains-webstorm-939

12. Bass L. – Software Architecture in Practice/ L. Bass – Addison-Wesley Professional, 2012

13. Anderson E. Software Engineering for Internet Applications / E. Anderson – MIT, 2005

14. Dykstra T. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application [Електронний ресурс] / T. Dykstra — Режим доступу: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

15. Lerman J. Data Points - Deny Table Access to the Entity Framework Without Causing a Mutiny [Електронний ресурс] / Lerman J. — Режим доступу: https://msdn.microsoft.com/en-us/magazine/ff898427.aspx

16. Singh A. What Is Architecture Overview of Angular? [Електронний ресурс] / Singh A. — Режим доступу: https://www.code-sample.com/2018/01/angular-4-and-5-architecture-overview.html

17 Stieven M. Angular: Understanding Modules and Services [Електронний ресурс] / M. Stieven — Режим доступу: https://medium.com/@michelestieven/organizing-angular-applications-f0510761d65a

18 Soper N. Building a RESTful API with ASP.NET 5 [Електронний ресурс] / N. Soper — Режим доступу: https://blog.scottlogic.com/2016/01/20/restful-api-with-aspnet50.html

19 Hassan Z. Managing Data With ViewModel In ASP.NET MVC [Електронний ресурс] / Z. Hassan — Режим доступу: https://www.c-sharpcorner.com/article/managing-data-with-viewmodel-in-asp-net-mvc/

20 Joudeh T. ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API [Електронний ресурс] — Режим доступу: http://bitoftech.net/2015/03/11/asp-net-identity-2-1-roles-based-authorization-authentication-asp-net-web-api/

21 Wasson M. Create Data Transfer Objects (DTOs) [Електронний ресурс] / M. Wasson — Режим доступу: https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

22 Kankar P. How To Implement Token Based Authentication Using Web API [Електронний ресурс] / P. Kankar — Режим доступу: https://www.c-sharpcorner.com/article/how-to-implement-token-based-authentication-using-webapi-entity-framework-and-a/

23 Vega D. Fluent API - Configuring and Mapping Properties and Types — [Електронний ресурс] / D. Vega — Режим доступу: https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/fluent/types-and-properties

24 Симан М. Внедрение зависимостей в .NET / М. Симон. — Питер, 2013 — 464 с.

25 Lerman L — Programming Entity Framework: Code First / L. Lerman — CreateSpace Independent Publishing Platform, 2009 — 164 с.

ДОДАТОК А

Система обліку та аналізу науково-дослідних робіт

Специфікація

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5149_19Б

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ" КПІ ім. Ігоря Сікорського"_ТЕФ_АПЕПС_ТР5149_19Б	Записка.doc	Пояснювальна записка
Компоненти		
УКР.НТУУ" КПІ ім. Ігоря Сікорського"_ТЕФ_АПЕПС_ТР5149_19Б 12-1	ContractController.cs	Контроллер для обробки запитів
УКР.НТУУ" КПІ ім. Ігоря Сікорського"_ТЕФ_АПЕПС_ТР5149_19Б 12-2	ContractService.cs	Сервіс для оперування даними
УКР.НТУУ" КПІ ім. Ігоря Сікорського"_ТЕФ_АПЕПС_ТР5149_19Б 12-3	Service.cs	Базовий набір методів для роботи з даними

ДОДАТОК Б

Система обліку та аналізу науково-дослідних робіт

Лістинг програмного модуля

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5149_19Б 12-2

Аркушів 8

Київ – 2019

-2-

```
namespace AccountingResearchWork.API.Controllers
{
    [ApiAuthorize] //окремо створена авторизація системи, з підтримкою ролей
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    [RoutePrefix("api/contract")]
    public class ContractController : BaseApiController
    {
        private readonly IClientDivisionService _clientDivisionService;
        private readonly IExecutiveDivisionService _executiveDivisionService;
        private readonly IStageService _stageService;
        private readonly IContractService _contractService;
        private readonly IMapper _mapper;

        public ContractController(IClientDivisionService
clientDivisionService, IExecutiveDivisionService executiveDivisionService,
                          IStageService stageService, IContractService
contractService, IMapper mapper)
        {
            _clientDivisionService = clientDivisionService;
            _executiveDivisionService = executiveDivisionService;
            _stageService = stageService;
            _contractService = contractService;
            _mapper = mapper;
        }

        [Route("{id}"), HttpGet]
        public virtual async Task<IHttpActionResult> Get(int id)
        {
            var model = _mapper.Map<ContractViewModel>(await
_contractService.GetAsync(id));

            if (model == null)
            {
                return NotFound();
            }

            return Ok(model);
        }

        [Route(""), HttpGet]
        public virtual async Task<IHttpActionResult> GetAll()
        {
            var models = _mapper.Map<IEnumerable<ContractViewModel>>(await
_contractService.GetAllAsync());

            if (!models.Any())
            {
                return NotFound();
            }

            return Ok(models);
        }

        [Route("analysis"), HttpGet]
        public virtual async Task<IHttpActionResult> GetAnalysisModels()
        {
```

```
            var models = await _contractService.GetAllAsync();

            if (!models.Any())
            {
                return NotFound();
            }

            return Ok(await MapToAnalysisModel(models));
        }

        [Route("find")]
        [HttpPost]
        public async Task<IHttpActionResult>
FindContracts([FromBody]ContractSearch search)
            => Ok((await _contractService.FindMentions(search)));

        [Route("")]
        [HttpPost]
        public async Task<IHttpActionResult> Post([FromBody]
ContractCreateViewModel contract)
        {
            if (contract == null) return BadRequest();
            var client = await
_clientDivisionService.GetAsync(contract.ClientDivisionClientId);
            var executor = await
_executiveDivisionService.GetAsync(contract.ExecutiveDivisionId);

            try
            {
                await _contractService.AddAsync(MapViewToDto(contract,
client, executor));
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }

            return Ok();
        }

        [Route("{id}")]
        [HttpPut]
        public async Task<IHttpActionResult> Put(long id, [FromBody]
ContractCreateViewModel contract)
        {
            if (contract == null) return BadRequest();
            contract.Id = id;

            var client = await
_clientDivisionService.GetAsync(contract.ClientDivisionClientId);
            var executor = await
_executiveDivisionService.GetAsync(contract.ExecutiveDivisionId);

            try
            {
```

```
                await _contractService.UpdateAsync(MapViewToDto(contract,
client, executor));
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }

            return Ok();
        }

        private static ContractDTO MapViewToDto(ContractCreateViewModel
createViewModel, ClientDivisionDTO client, ExecutiveDivisionDTO executor)
        {
            var result = new ContractDTO
            {
                Name = createViewModel.Name,
                BeginningDate = createViewModel.BeginningDate,
                CompletionDate = createViewModel.CompletionDate,
                Client = client,
                Executor = executor
            };

            if (createViewModel.Id.HasValue)
            {
                result.Id = createViewModel.Id.Value;
            }

            return result;
        }



namespace AccountingResearchWork.BL.Services
{
    public class ContractService : Service<Contract, ContractDTO>,
IContractService
    {
        public ContractService(IUnitOfWork database, IContractRepository
repository, IMapper mapper)
            : base(database, repository, mapper)
        {
        }

        public async Task<IEnumerable<ContractDTO>>
FindMentions(ContractSearch searchModel)
        {
            var result = await this.GetAllAsync();

            if (!string.IsNullOrWhiteSpace(searchModel.Name))
                result = result.Where(x =>
x.Name.Contains(searchModel.Name));

            if (searchModel.BeginningDate.HasValue)
```

```
                result = result.Where(x => x.BeginningDate <
searchModel.BeginningDate);
            if (searchModel.CompletionDate.HasValue)
                result = result.Where(x => x.CompletionDate <
searchModel.CompletionDate);

            if (!string.IsNullOrWhiteSpace(searchModel.ClientDivisionName))
                result = result.Where(x =>
x.Client.Name.Contains(searchModel.ContractorDivisionName));

            if
(!string.IsNullOrWhiteSpace(searchModel.ContractorDivisionName))
                result = result.Where(x =>
x.Executor.ContractorDivision.Name.Contains(searchModel.ContractorDivisionNam
e));
            if (!string.IsNullOrWhiteSpace(searchModel.OrganizationName))
                result = result.Where(x =>
x.Executor.ContractorDivision.Organization.Name.Contains(searchModel.Organiza
tionName));

            return result;
        }
    }
}



namespace AccountingResearchWork.BL.Services.Base
{
    public class Service<TEntity, TEntityDto> : IService<TEntityDto>
        where TEntityDto : class
        where TEntity : class
    {
        protected IUnitOfWork Database { get; }
        protected IRepository<TEntity> Repository { get; }
        protected IMapper Mapper { get; }

        public Service(IUnitOfWork database, IRepository<TEntity> repository,
IMapper mapper)
        {
            Database = database;
            Repository = repository;
            Mapper = mapper;
        }

        public virtual TEntityDto Get(long id)
        {
            return Mapper.Map<TEntityDto>(Repository.Get(id));
        }

        public virtual async Task<TEntityDto> GetAsync(long id)
            => Mapper.Map<TEntityDto>(await Repository.GetAsync(id));

        public virtual IEnumerable<TEntityDto> GetAll()
            => Mapper.Map<IEnumerable<TEntityDto>>(Repository.GetAll());
```

```
        public virtual async Task<IEnumerable<TEntityDto>> GetAllAsync()
            => Mapper.Map<IEnumerable<TEntityDto>>(await
Repository.GetAllAsync());

        public virtual void Add(TEntityDto entityDto)
        {
            if (entityDto == null) throw new
ArgumentNullException(nameof(entityDto));

            var entity = Mapper.Map<TEntity>(entityDto);

            Repository.Add(entity);
            Database.Save();
        }

        public async Task AddAsync(TEntityDto entityDto)
        {
            if (entityDto == null) throw new
ArgumentNullException(nameof(entityDto));

            var entity = Mapper.Map<TEntity>(entityDto);

            Repository.Add(entity);
            try
            {
                await Database.SaveAsync();
            }
            catch (Exception e)
            {
            }
        }

        public virtual void AddRange(IEnumerable<TEntityDto> entitiesDto)
        {
            if (entitiesDto == null) throw new
ArgumentNullException(nameof(entitiesDto));
            var entities = Mapper.Map<IEnumerable<TEntity>>(entitiesDto);

            Repository.AddRange(entities);
            Database.Save();
        }

        public async Task AddRangeAsync(IEnumerable<TEntityDto> entitiesDto)
        {
            if (entitiesDto == null) throw new
ArgumentNullException(nameof(entitiesDto));
            var entities = Mapper.Map<IEnumerable<TEntity>>(entitiesDto);

            Repository.AddRange(entities);
            await Database.SaveAsync();
        }

        public virtual void Remove(TEntityDto entityDto)
        {
            if (entityDto == null) throw new
ArgumentNullException(nameof(entityDto));
```

```
            var entity = Mapper.Map<TEntity>(entityDto);

            Repository.Remove(entity);
            Database.Save();
        }

        public async Task RemoveAsync(TEntityDto entityDto)
        {
            if (entityDto == null) throw new
ArgumentNullException(nameof(entityDto));
            var entity = Mapper.Map<TEntity>(entityDto);

            Repository.Remove(entity);
            await Database.SaveAsync();
        }

        public virtual void RemoveRange(IEnumerable<TEntityDto> entitiesDto)
        {
            if (entitiesDto == null) throw new
ArgumentNullException(nameof(entitiesDto));
            var entities = Mapper.Map<IEnumerable<TEntity>>(entitiesDto);

            Repository.RemoveRange(entities);
            Database.Save();
        }

        public async Task RemoveRangeAsync(IEnumerable<TEntityDto>
entitiesDto)
        {
            if (entitiesDto == null) throw new
ArgumentNullException(nameof(entitiesDto));
            var entities = Mapper.Map<IEnumerable<TEntity>>(entitiesDto);

            Repository.RemoveRange(entities);
            await Database.SaveAsync();
        }

        public virtual void Update(TEntityDto entityDto)
        {
            if (entityDto == null) throw new
ArgumentNullException(nameof(entityDto));
            var entity = Mapper.Map<TEntity>(entityDto);

            Repository.Update(entity);
            Database.Save();
        }

        public async Task UpdateAsync(TEntityDto entityDto)
        {
            if (entityDto == null) throw new
ArgumentNullException(nameof(entityDto));
            var entity = Mapper.Map<TEntity>(entityDto);

            Repository.Update(entity);
            await Database.SaveAsync();
        }
```

```
        public void Dispose()
            => Database.Dispose();
    }
}



namespace AccountingResearchWork.DAL.Domain
{
    public class Contract : Base
    {
        public string Name { get; set; }

        public DateTime BeginningDate { get; set; }

        public DateTime CompletionDate { get; set; }

        public ExecutiveDivision Executor { get; set; }

        public ClientDivision Client { get; set; }
    }
}
```

ДОДАТОК В

Система обліку та аналізу науково-дослідних робіт

Опис програмного модуля

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТР5149_19Б 12-2

Аркушів 8

Київ – 2019

ЗМІСТ

1. Загальні відомості 3
2. Функціональне призначення 4
3. Опис логічної структури 5
4. Технічні засоби, що використовувалися 6
5. Вхідні та вихідні дані 7

-3-

ЗАГАЛЬНІ ВІДОМОСТІ

Програмний код включає в себе частину архітектури системи, що використовується для обробки запиту клієнту.

Запит оброблюється відповідними методами класу ContractController, який також гарантує авторизований доступ до даних.

Для отримання / редагування даних метод контроллеру звертається до відповідного сервісу.

Веб-додаток реалізовано як API за допомогою WebAPI та використовуючи REST для задання шаблону звертань до даних.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмний продукт надає такі функціональні можливості:

— веб-інтерфейс для створення даних;

— генерація результату аналізу введених в систему даних;

— аналітичні дані виводяться в таблицю в браузері;

-5-

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Робота з системою здійснюється за принципом клієнт-сервера

Запита з клієнта за допомогою системи маршрутизації опрацьовуються за вказаним URL та HTTP-методом. Для кожної операції, що виконується клієнтом є відповідний метод контроллера. Для того, щоб додати запис, клієнт надсилає запит POST з відповідною моделлю на URL “api/contract”, після отримання запиту викликається метод Post контроллера. Для обробки запиту викликається метод Add відповідного сервісу для договорів, що додає запис за надісланою клієнтом моделлю в БД.

-6-

ТЕХНІЧНІ ЗАСОБИ, ЩО ВИКОРИСТОВУВАЛИСЯ

Систему було створено мовою програмування C# у середовищі Visual Studio 2017 для серверної частини та мовою TypeScript та WebStorm для клієнту.

Для створення API було використано WebAPI, для візуалізації даних використовуються HTML, CSS та Bootstrap.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідними даними є:

— назва договору

— дата початку роботи над договором;

— дата планованого закінчення робіт;

— організація замовника (назва, відповідальні особи);

— виконавець (організація, підрозділ, виконавець, науковий керівник).

Вихідними даними є:

— компоненти Angular, що візуалізують відповідь сервера;